



ДИАЛОГ-МИСОН

**БИБЛИОТЕКА
СИСТЕМНОГО
ПРОГРАММИСТА**

**ОПЕРАЦИОННАЯ
СИСТЕМА MS-DOS**

Том 1
ИНВ. КН.

1-2

БИБЛИОТЕКА СИСТЕМНОГО ПРОГРАММИСТА

А. В. ФРОЛОВ, Г. В. ФРОЛОВ

ОПЕРАЦИОННАЯ СИСТЕМА MS-DOS

ТОМ **1**
КНИГИ 1, 2

ИЗДАНИЕ ВТОРОЕ, С ИЗМЕНЕНИЯМИ

МОСКВА • "ДИАЛОГ-МИФИ" • 1992

ББК 32.973.1
Ф91

БИБЛИОТЕКА
СИСТЕМНОГО ПРОГРАММИСТА
Выпускается с 1991 года

Фролов А.В., Фролов Г.В.

Ф91 Операционная система MS-DOS: В 3 кн. Кн. 1-2.—2-е изд.,
с изменен.—М.: "ДИАЛОГ-МИФИ", 1991.—240 с.—(Библи-
отека системного программиста; Т. 1, Кн. 1-2)
ISBN 5-86404-020-7 (Т. 1, Кн. 1-2)

Оригинальное учебно-справочное руководство по системному программированию в операционной системе MS-DOS на компьютерах IBM PC. От аналогичных изданий отличается более глубоким изложением материала. Описываются детали, которые часто остаются скрытыми даже для опытного программиста: внутренняя структура и организация работы MS-DOS, недокументированные прерывания. Разбираются профессиональные приемы работы. Приводится большое количество подробно комментируемых исходных текстов программ.

Для системных программистов, аспирантов, студентов вузов.

Ф 2404090000-004
Г70(03)-92

Без объявл.

ББК 32.973.1

Учебно-справочное издание

Фролов Александр Вячеславович

Фролов Григорий Вячеславович

ОПЕРАЦИОННАЯ СИСТЕМА MS-DOS

В 3 книгах, книги 1 и 2

Научные редакторы: С.А. Гладков, Е.А. Устинов
Художественный редактор О.А. Кузьмина
Технический редактор О.А. Красильникова
Корректор В.С. Кустов

Подписано в печать 11.02.92. Усл. печ. л. 13,95. Уч.-изд. л. 12.
Тираж 4 000 экз. Заказ 341

Акционерное общество "ДИАЛОГ-МИФИ"
115409 Москва, ул.Москворечье, 31, корп. 2

Типография Московского инженерно-физического института
115409 Москва, Каширское ш., 31

ISBN 5-86404-020-7 (Т. 1, Кн. 1-2)

ISBN 5-86404-004-5

© А.В. Фролов, Г.В. Фролов, 1991

© Оригинал-макет, оформление обложки.
АО "ДИАЛОГ-МИФИ", 1992

Добро пожаловать в мир системного программирования! Перед Вами первый том "Библиотеки системного программиста", которую начинает издавать акционерное общество "ДИАЛОГ-МИФИ". В этой серии мы планируем выпускать книги, рассчитанные как на начинающих системных программистов, так и тех, кто уже давно работает в этой области.

К настоящему времени подготовлены первые 4 тома "Библиотеки", посвященные, главным образом, операционной системе MS-DOS и аппаратному обеспечению компьютера IBM PC. В будущем планируется выпустить тома, посвященные другим операционным системам (OS/2, UNIX и т.п.), а также новым типам периферийного оборудования: лазерные диски, сканеры, стриммеры...

Редакционно-издательский отдел
АО "ДИАЛОГ-МИФИ"

ПРЕДИСЛОВИЕ АВТОРОВ

В настоящее время можно найти уже довольно много книг, посвященных операционной системе MS-DOS. Большинство из них, однако, ограничиваются описанием MS-DOS на уровне пользователя или, в крайнем случае, на уровне прикладного программиста, не затрагивая деталей и тонкостей работы самой операционной системы. Этот подход, безусловно, правомерен и оправдан - пользователей ПЭВМ гораздо больше, чем системных программистов, а более глубокая информация содержится в фирменных руководствах и специальных заказных пособиях.

И все же потребность в казалась бы специальной информации у нас огромна. Советский программист часто оказывается в очень сложной ситуации - не имея доступа к зарубежной оригинальной литературе он вынужден разрабатывать специальное программное обеспечение, написание которого требует глубокого знания операционной системы MS-DOS и аппаратных особенностей IBM PC. Можно с уверенностью сказать, что в условиях информационного голода, отсутствия фирменного технического обслуживания и необходимости обеспечения работы самой экзотической аппаратуры каждый прикладной программист в СССР вынужден быть немножко системным программистом, знать и уметь больше, чем его западный коллега.

Как известно, спрос рождает предложение, и уже сейчас появились книги под названием "Системное программирование в MS-DOS", в которых описываются, как правило, прерывания MS-DOS и приводятся примеры работы с ними. Такой подход нам кажется полезным только на начальном этапе изучения программирования в MS-DOS.

При подготовке материала по MS-DOS для "Библиотеки" авторы предполагали, что с прерываниями DOS и BIOS вы уже знакомы достаточно хорошо либо способны разобраться самостоятельно - для этого существуют как специальные справочные системы типа TechHelp (которая сейчас имеется и на русском языке) или Norton Guide, так и контекстные справочники, являющиеся элементом интегрированных сред программирования (например, QuickAdvisor корпорации Microsoft или справочная система фирмы Borland International).

Авторы стремились прежде всего осветить те вопросы, которые практически невозможно найти нигде, кроме фирменной документации. В соответствии с этим общеизвестные вещи изложены более кратко. Например, описана структура управляющих блоков MS-DOS, показано, как написать собственный драйвер устройства или правильно работающую резидентную программу. Материал рассчитан на хорошего прикладного программиста, имеющего в своем распоряжении широко распространенную справочную информацию по MS-DOS, который, однако, пришел к необходимости еще более углубить свои знания.

Предполагается, что у вас имеется в распоряжении компьютер, на котором вы в процессе изучения можете опробовать приводимые нами примеры и проводить свои собственные исследования.

Все программы транслировались в среде Microsoft QuickC версий 2.01 и 2.5. Примеры составлены так, чтобы вы могли без значительных переделок использовать их в своих разработках.

Для тех, кто уже исчерпал документированные особенности MS-DOS, приводятся сведения о наиболее полезных недокументированных прерываниях и управляющих блоках MS-DOS - изучение недокументированных прерываний и структур данных позволит вам глубже понять внутреннее устройство операционной системы, извлечь такую информацию о состоянии системы, которую трудно, если вообще возможно, получить "законным" способом.

Материал по MS-DOS изложен в нескольких томах.

Первый том серии содержит сведения об операционной системе MS-DOS, прерываниях, драйверах, резидентных программах, файловой системе MS-DOS. Основное внимание при этом уделяется не таким общеизвестным вещам, как, например, запись/чтение файлов (хотя об этом тоже будет рассказано), а скорее описанию того, как MS-DOS выполняет эти операции, какие внутренние структуры данных она при этом использует и каким образом происходит взаимодействие DOS и прикладной программы пользователя.

На основе анализа управляющих блоков DOS вы сможете проанализировать состояние операционной системы, определить конфигурацию логических дисковых устройств, получить доступ к загружаемым (устанавливаемым) драйверам, в том числе к резидентным драйверам опе-

рационной системы. Вы научитесь пересопределять прерывания DOS, составлять "выскакивающие" ("POP-UP") программы, которые, оставаясь резидентными в памяти после запуска, "оживают" при нажатии на заранее определенную клавишу. Знание внутренней "кухни" файловой системы позволит вам при необходимости выполнять все файловые операции самостоятельно, без помощи DOS, пользуясь только прерыванием BIOS INT 13H для непосредственной работы с диском (такая работа в обход DOS может потребоваться, например, для организации защиты данных от несанкционированного доступа).

Второй том посвящен в большей степени аппаратному обеспечению компьютера. Здесь описаны клавиатура, мышь, таймер, принтер и другое оборудование персонального компьютера. Приведены сведения о внутреннем устройстве клавиатуры, портах, средствах DOS и BIOS для работы с клавиатурой, т. е. все, что необходимо для полного использования возможностей клавиатуры. Для мыши приводится большое число примеров программ, которые вы можете использовать в своих разработках.

Подробно описаны порты принтера, средства DOS и BIOS для работы с принтером, а также система принтерного спулинга (печать в фоновом режиме). Рассматриваются вопросы русификации принтеров и управления принтером, поддерживающим протокол фирмы Epson.

Третий том - это описание дисплейных адаптеров. Рассмотрены адаптеры CGA, EGA и VGA. Приводится описание видеорежимов, средств DOS и BIOS для управления адаптерами и вывода текстовой и графической информации, способов загрузки знакогенератора адаптера EGA, а также основных возможностей графической библиотеки Microsoft QuickC 2.5.

Четвертый том содержит сведения об использовании инструментальных средств разработки программного обеспечения - интегрированных систем Microsoft QuickC 2.01 и 2.5, пакетном трансляторе Microsoft C 6.0. Описаны средства отладки программного обеспечения - Microsoft CodeView 3.00 и отладчик фирмы IBM AFD (Advanced Fullscreen Debug).

Приведены рекомендации по отладке программ с использованием этих средств, по отладке программ специального типа, например драйверов, а также основные приемы, используемые для защиты программ от отладки.

Как пользоваться книгой?

Можно просто читать текст, изучая приводимые примеры программ и тут же проверяя их работу на компьютере. Все сведения излагаются последовательно, поэтому вы сразу можете садиться за компьютер и начинать работать. Если вас интересует что-то конкретное, например драйверы, вы можете начать сразу с соответствующей главы.

Прилагаемые дискеты^{*)} содержат не только исходные тексты примеров программ, но и готовые библиотеки объектных модулей для всех моделей памяти, а также справочную базу данных по этим модулям. Базу данных можно подключить к справочной базе интегрированной среды QuickC или использовать отдельно при помощи утилиты Microsoft Quick Help QH.EXE.

Все необходимые сведения об использовании содержимого дискет приведены в файле README.DOC.

Каков необходимый начальный уровень знаний? Предполагается, что вы свободно владеете языками Ассемблера и Си (хотя в некоторых случаях приводятся необходимые пояснения), умеете пользоваться стандартными прерываниями BIOS и DOS, знакомы в целом с архитектурой компьютера и имеете некоторый опыт составления программ.

В самом начале мы напомним вам о составе операционной системы, процессе ее загрузки и об общей схеме работы. Первый раздел содержит обзор прерываний DOS и BIOS, сведения о механизме обработки ошибок. Если вы владеете этим материалом, можете его пропустить.

Для работы вы можете использовать любой совместимый с IBM PC/XT/AT компьютер с любым дисплейным адаптером. Однако при изучении дисплейных адаптеров EGA и VGA вам будет нужен соответственно адаптер EGA или VGA. Желательно, чтобы компьютер был оснащен жестким диском (на машине, например, ЕС-1840, вам будет очень трудно работать). Наличие жесткого диска обязательно для изучения глав, посвященных файловой системе. Для изучения мыши вам следует приобрести это устройство.

^{*)} Дискеты можно приобрести в киоске "ДИАЛОГ-МИФИ" по адресу: Москва, ул. Москворечье, 31, корп. 2. Справки по телефону 324-83-56

Все приведенные программы подготовлены для Microsoft QuickC или Microsoft C 6.0. Не исключено, что вы сможете использовать Turbo C фирмы Borland, если приведете программы в соответствие со стандартами Turbo C.

Программы, составленные на языке Ассемблера, транслировались при помощи программы QuickAssembler, входящей в состав интегрированной среды QuickC 2.01. Возможно также использование ассемблера MASM версии 5.0 или более поздней версии, программы Turbo Assembler фирмы Borland с учетом приведенных выше замечаний.

Еще одно замечание.

Системное программирование - это, исходя из названия, программирование системных задач, создание операционных систем или отдельных компонентов операционных систем, таких, например, как драйверы внешних устройств.

В отличие от прикладного программиста системный программист должен в одинаковой степени владеть и программным, и аппаратным обеспечением компьютера. Если прикладной программист, как правило, не работает напрямую с аппаратурой, пользуясь сервисом операционной системы, то одна из основных задач системного программиста - организация обслуживания устройств ввода/вывода.

Поэтому одинаковое внимание будет уделяться как вопросам использования операционной системы, так и особенностям работы аппаратуры.

Однако при создании драйвера устройства вам следует ориентироваться прежде всего на фирменное техническое описание устройства - только там приводятся все технические подробности, без учета которых ваш драйвер не будет правильно работать.

Очень осторожно следует использовать недокументированные средства операционной системы: может получиться так, что в другой версии операционной системы ваша отлаженная программа будет делать совсем не то, для чего она предназначена.

Авторы выражают признательность А.М.Болотовой как первому читателю и редактору книги, С.А.Гладкобу, высказавшему много ценных советов по структуре материала и его оформлению, всем сотрудникам АО "ДИАЛОГ-МИФИ", принимавшим участие в работе над книгой.

А.В.Фролов, Г.В.Фролов

ГЛАВА 1

ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ MS-DOS

1.1. Основные компоненты операционной системы

Дистрибутив операционной системы MS-DOS состоит, в зависимости от версии, из одной или нескольких дискет. На них расположены файлы собственно операционной системы IO.SYS, MSDOS.SYS, командный процессор COMMAND.COM, файлы внешних команд операционной системы (FORMAT, FDISK и т. п.), драйверы и другие файлы.

Файл IO.SYS содержит расширение базовой системы ввода/вывода и является интерфейсом между операционной системой и BIOS. Расширение используется операционной системой для взаимодействия с аппаратурой компьютера и BIOS.

Файл MSDOS.SYS является в некотором смысле набором программ обработки прерываний, в частности прерывания INT 21H. Это тело операционной системы.

Командный процессор COMMAND.COM предназначен для организации диалога с оператором. Он анализирует вводимые оператором команды и организует их выполнение. Так называемые внутренние команды (DIR, COPY и т. д.) обрабатываются именно командным процессором. Программист имеет возможность написать свой собственный командный процессор и подключить его вместо стандартного. Новый командный процессор должен выполнять все функции, которые раньше выполнял стандартный COMMAND.COM.

Драйверы (обычно это файлы, имеющие расширение имени .SYS) представляют собой программы, обслуживающие различную аппаратуру. Эти программы имеют специальный формат и будут подробно описаны в книге. Применение драйверов легко решает проблемы использования новой аппаратуры - достаточно написать для нового устройства драйвер и подключить его к операционной системе. Прикладные программы взаимодействуют с устройствами через драйвер, поэтому они не будут меняться при изменениях в аппаратуре. Например, новое дисковое устройство может иметь другое количество дорожек и секторов, другие управляющие команды. Все это учитывается драйвером, а прикладная программа будет работать с новым диском, как и раньше, используя прерывания DOS.

Файлы внешних команд операционной системы содержат программы-утилиты для выполнения разнообразных операций, таких, как форматирование дисков, сортировка файлов, печати текстов и др.

Немного об утилитах, предназначенных для подготовки дисков и дискет.

Файлы операционной системы выделяются своим особым расположением на диске (кроме COMMAND.COM) - эти файлы должны находиться в специально отведенном для них месте. Если вам нужно подготовить дискету как системную (т. е. такую, с которой можно загружать операционную систему), для переноса файлов операционной системы следует использовать специальные утилиты.

Самый простой способ подготовки системной дискеты - использовать команду **FORMAT** с опцией **/S**, например:

```
FORMAT A: /S
```

В этом случае после форматирования на дискету будут перенесены файлы операционной системы. При использовании команды текущим должен быть корневой каталог системного диска, например диска **C:**.

Если вы собираетесь обновить версию операционной системы (например, вместо **MS-DOS 3.30** установить **MS-DOS 4.01**), не обязательно заново переформатировать весь диск. Загрузив новую версию с дискеты, для переноса новых системных файлов используйте команду **SYS**:

```
SYS C:
```

Файл **COMMAND.COM** скопируйте обычным способом.

Если вам надо сделать дискету системной, а ее форматирование нежелательно (дискета содержит важную информацию), воспользуйтесь программой **Norton Disk Doctor** или аналогичной. Программа освободит место в начале диска для операционной системы, переписав располагавшиеся там данные на свободное место в конце дискеты, затем запишет системные файлы и даже скопирует файл **COMMAND.COM**.

Утилита **FDISK** предназначена для подготовки к работе жесткого диска. Она разбивает диск на участки, называемые разделами.

На одном физическом диске могут быть разделы, принадлежащие разным операционным системам. Один из разделов - активный, операционная система загружается из активного раздела.

Для **DOS** утилита **FDISK** позволяет создать первичный и вторичный разделы. В первичном разделе располагается системный диск **C:**, с которого выполняется загрузка операционной системы, вторичный раздел может быть разделен на логические диски (**D**, **E**, **F** и т. д.). Диски, располагающиеся во вторичном разделе, не могут быть системными.

Заметим, что только **MS-DOS** версии **4.01** и **Compaq DOS 3.31** позволяют создавать логические диски размером более **32 Мбайт**. Это связано с тем, что другие версии **DOS** используют 16-разрядную адресацию секторов диска, что недостаточно для дисков больших размеров.

Очень часто вместо утилиты **FDISK** для подготовки жесткого диска используются диск-менеджеры. Это такие программы, как **Advanced Disk Manager**, **Speed Stor** и т. д. Используя свои собственные форматы разделов и таблиц разделов (и свои драйверы дисковых устройств), диск-менеджеры предоставляют такие дополнительные возможности, как защита логического диска от записи или организация парольной защиты данных на диске, создание логических дисков размером более **32 Мбайт**.

Однако не всегда применение диск-менеджеров может привести к желаемому результату. Защита от несанкционированного доступа часто легко преодолевается, мощные системы защиты сильно привязаны к конкретной версии операцион-

ной системы (например, WatchDog, очень мощная система защиты, требует только MS-DOS версии 3.20).

Некоторые программы, особенно защищенные от копирования, отказываются работать на диске, подготовленном не утилитой FDISK. Причины этого мы увидим, когда будем обсуждать проблемы защиты программ от несанкционированного копирования.

Перейдем к процедуре начальной загрузки операционной системы.

1.2. Процесс загрузки операционной системы

При включении питания компьютера управление передается базовой системе ввода/вывода, BIOS. Она выполняет проверку аппаратных узлов компьютера, формирует начальную часть таблицы векторов прерываний, инициализирует устройства и начинает процесс загрузки операционной системы.

Загрузка начинается с того, что BIOS делает попытку прочитать самый первый сектор дискеты, вставленной в дисковод A (на загрузочной дискете этот сектор содержит загрузчик операционной системы). Если в дисковод вставлена системная дискета, с нее считывается загрузчик и ему передается управление.

Если дискета не системная, т. е. не содержит загрузочной записи, на экран выдается сообщение с просьбой заменить дискету.

Если же дискеты в дисковом A вообще нет, то BIOS читает основную загрузочную запись диска C (Master Boot Record). Обычно это самый первый сектор на диске. Управление передается загрузчику, который находится в этом секторе. Загрузчик анализирует содержимое таблицы разделов (она также находится в этом секторе), выбирает активный раздел и читает загрузочную запись этого раздела. Загрузочная запись активного раздела (Boot Record) аналогична загрузочной записи, находящейся в первом секторе системной дискеты.

Загрузочная запись активного раздела считывает с диска файлы IO.SYS и MSDOS.SYS (именно в этом порядке). Затем считываются и загружаются резидентные драйверы. Начинается формирование связанного списка драйверов устройств. Анализируется содержимое файла CONFIG.SYS, загружаются описанные в этом файле драйверы. Сначала загружаются драйверы, описанные параметром DEVICE, затем (только в MS-DOS версии 4.x) резидентные программы, указанные операторами INSTALL. После этого считывается командный процессор и ему передается управление.

Командный процессор состоит из трех частей - резидентной, инициализирующей и транзитной. Первой загружается резидентная часть. Она обрабатывает прерывания INT 22H, INT 23H, INT 24H, управляет загрузкой транзитной части. Эта часть командного процессора обрабатывает ошибки MS-DOS и выдает запрос пользователю о действиях при обнаружении ошибок.

Инициализирующая часть используется только в процессе загрузки операционной системы. Она определяет начальный адрес, по которому будет загружаться пользовательская программа и инициализирует выполнение файла AUTOEXEC.BAT.

Транзитная часть командного процессора располагается в старших адресах памяти. В этой части находятся обработчики внутренних команд DOS и интерпретатор командных файлов с расширением имени .BAT. Транзитная часть выдает системное приглашение (например, A:\>), ожидает ввода команды оператора с клавиатуры или из пакетного файла и организует их выполнение.

После загрузки командного процессора и выполнения начальных процедур, перечисленных в файле AUTOEXEC.BAT, подготовка системы к работе завершается.

1.3. Общая схема работы DOS

Для того чтобы правильно работать с системным программным и аппаратным обеспечением, нужно четко представлять себе механизм взаимодействия прикладной программы с компьютером. На рисунке ниже показаны функциональные связи программы с программно-аппаратным обеспечением IBM PC:

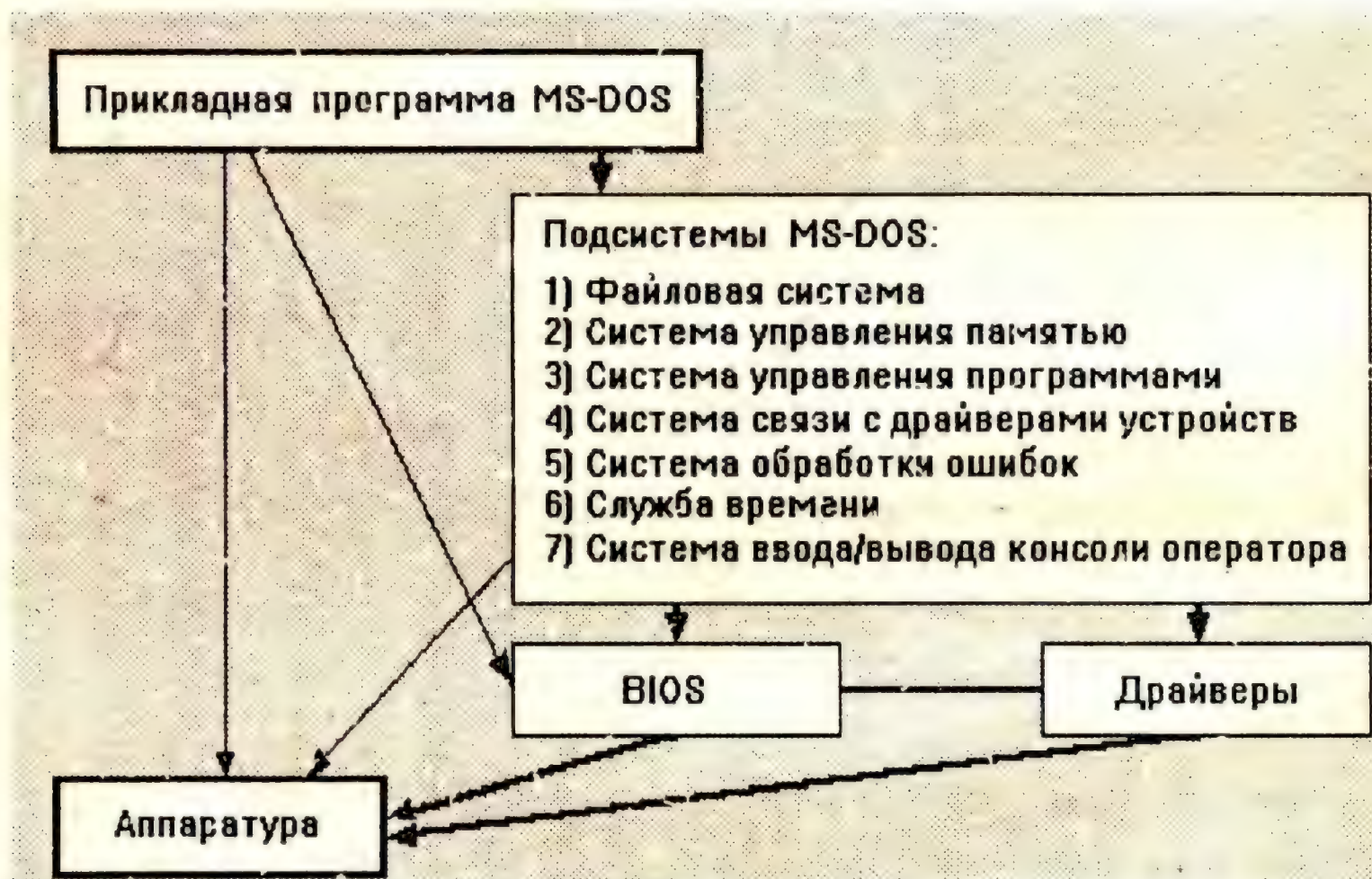


Рис.1. Функциональные связи программы для MS-DOS с программно-аппаратным обеспечением ПЭВМ

Как правило, ядро DOS разделяют на несколько подсистем, каждая из которых отвечает за выполнение той или иной задачи. Как показано на рисунке, обычно выделяются следующие подсистемы:

- файловая система
- система управления памятью
- система управления программами

- система связи с драйверами устройств
- система обработки ошибок
- служба времени
- система ввода/вывода консоли оператора

Эти подсистемы общаются с аппаратурой через BIOS, драйверы или напрямую. Прикладное программное обеспечение может вызывать подсистемы DOS, работать с BIOS или непосредственно с аппаратурой. Обратите, однако, внимание на то, что прикладные программы могут обращаться к драйверам только через соответствующую подсистему DOS.

Очевидно также, что чем выше уровень интерфейса прикладной программы и аппаратуры, тем меньше программа будет зависеть от особенностей аппаратуры.

Рассмотрим подсистемы DOS отдельно.

Файловая система

Файловая система является одной из важнейших подсистем DOS. Она используется как в процессе загрузки операционной системы, так и в процессе ее работы. Сама операционная система записана на системном диске в виде файлов (IO.SYS, MSDOS.SYS, COMMAND.COM, драйверы, внешние команды и т. д.). Все прикладные программы и вообще все программы и данные для них хранятся на дисках в виде файлов, поэтому можно сказать, что файловая система - ключевая подсистема DOS.

Для обращения к файловой системе прикладная программа должна использовать специально предназначенные для этого функции прерывания DOS. Эти функции выполняют все файловые операции: создание, удаление файлов и каталогов, буферизованная и небуферизованная запись или чтение, получение справочной информации о состоянии файловой системы и другие. Файловые операции будут кратко описаны в обзоре прерываний DOS, кроме того, файловой системе полностью посвящена третья книга первого тома.

Файловая система работает с дисками через драйверы. Драйверы, в свою очередь, пользуются сервисом BIOS.

На уровне BIOS выполняются элементарные операции с диском, такие, как чтение/запись секторов, форматирование и т. п. Этот низкий уровень доступен и прикладной программе, но обычно она пользуется функциями прерывания DOS, выполняющими все необходимые действия по обслуживанию каталогов и таблицы размещения файлов (File Allocation Table - FAT). Программы защиты от несанкционированного доступа или копирования вынуждены обращаться к средствам более низкого уровня, вызывая прерывания BIOS, или даже работать с контроллером дисководов через порты ввода/вывода.

Используя подсистему DOS, отвечающую за связь с драйверами, прикладные программы могут также выполнять элементарные операции с диском, такую, как форматирование.

Система управления памятью

Эта подсистема DOS используется для распределения памяти запускаемым программам.

DOS управляет памятью с помощью блоков MCB (Memory Control Block). Память разбивается на блоки; каждому блоку предшествует MCB, в котором записаны характеристики блока памяти. Для каждой вновь запускаемой программы DOS создает определенное количество блоков MCB. При освобождении памяти или при выполнении запросов на получение дополнительной памяти DOS также использует блоки MCB, проверяя при этом правильность их содержимого.

Все блоки MCB располагаются друг за другом. Адрес первого блока хранится в векторной таблице связи, CVT, о которой мы будем говорить в главе 2. Там же будет описан формат блока управления памятью.

Прикладная программа может заказать для себя дополнительные блоки памяти. Для этого она обращается к системе управления памятью, используя функции прерывания 21h DOS.

Система управления программами

При запуске программы DOS выполняет несколько операций. Сначала она обращается к системе управления памятью, чтобы подготовить блоки памяти для запускаемой программы, с помощью файловой системы файл, содержащий программу, загружается в память, после чего программа (это относится только к файлам типа .exe) настраивается на конкретный физический адрес. Только после этого программе передается управление.

Как известно, в MS-DOS существуют два формата выполняемых программ - .com и .exe. Технологии запуска этих программ сильно различаются. Система управления программами автоматически распознает их и загружает в память по-разному. Мы еще вернемся к описанию различий между этими типами программ.

Другая задача, решаемая системой управления программами, - запуск программ из программ и загрузка "программных перекрытий" - оверлеев. Если не все модули большого программного комплекса нужны одновременно, вы можете разбить комплекс на несколько частей. Это могут быть либо несколько отдельных программ, либо несколько оверлейных модулей. Каждый из этих способов имеет свои преимущества и недостатки; оба они пригодны для экономии памяти.

И, наконец, последняя функция системы управления программами - работа с резидентными программами. Если вам надо, чтобы после завершения своей работы программа осталась резидентной в памяти, вы, как и в случае завершения обычной программы, обращаетесь к системе управления программами через соответствующую функцию прерывания DOS 21h.

Система связи с драйверами устройств

Эта система скрыта от прикладных программ - программы не могут обращаться непосредственно к драйверам устройств ввода/вывода. Программа вызывает DOS, а DOS обращается при необходимости к драйверам.

Возможно, что запрет на непосредственный вызов драйверов введен для обеспечения совместимости с будущими версиями операционной системы, в которых механизм вызова драйверов может измениться. Однако, используя сведения, приведенные в этой книге, вы сможете обойти этот запрет и обратиться непосредственно к драйверу. При этом вам придется использовать некоторые недокументированные прерывания DOS, что само по себе нежелательно из-за возможной потери совместимости с другими версиями операционной системы.

Для управления состоянием устройства ввода/вывода или состоянием драйвера используется специальная функция 44h прерывания DOS 21h. Эта функция предназначена для обмена управляющей информацией между прикладной программой и драйвером.

Система обработки ошибок

Система обработки ошибок DOS проста и удобна. Для кодирования ошибок, как правило, используется флаг переноса (CARRY, CF). Если после обращения к прерыванию DOS флаг переноса установлен в 1, произошла ошибка. Для того, чтобы проанализировать ошибку и предпринять какие-то действия, можно вызвать соответствующую функцию DOS, которая вернет уточняющую информацию об ошибке и предоставит соответствующие рекомендации (разумеется, лишь в виде кодов, находящихся в регистрах процессора).

Если произошла критическая ошибка ввода/вывода (например, прочитать диску невозможно), вызывается стандартная процедура DOS, выводящая на экран запрос о дальнейших действиях. Пользовательская программа может подключить вместо системной свою программу обработки критических ошибок. Подробнее об обработке ошибок будет сказано в разделе 1.6.

Служба времени

Компьютер обычно оборудуется системными часами. Это могут быть КМОП-часы с питанием от аккумулятора, содержимое которых не сбрасывается при выключении питания компьютера, или таймер, регулярно вырабатывающий прерывания. В любом случае операционная система ведет подсчет времени и хранит текущие показания часов и дату.

Программа может опросить часы, обратившись к DOS с запросом через одну из функций прерывания 21h, или установить новое состояние часов.

Операционная система содержит драйвер устройства CLOCK\$. Прикладная программа может обратиться к этому устройству для чтения показания часов или для установки часов. В книге 2 первого тома приведен пример программы для работы с устройством CLOCK\$.

Программа может также использовать прерывания таймера для регулярного выполнения каких-либо функций.

Ввод/вывод на консоль оператора

Консоль оператора состоит из двух устройств - клавиатуры и дисплея. Эти два устройства обслуживаются одним драйвером - драйвером консоли CON. Т.е. можно считать, что в компьютере имеется устройство - консоль - с именем CON.

Операционная система обслуживает консоль с помощью функций прерывания 21h, обеспечивающих ввод и вывод символов на устройство CON. Для работы с физической клавиатурой и дисплейным адаптером этот драйвер использует прерывания BIOS.

Другие подсистемы DOS

DOS имеет набор драйверов для работы с печатающими устройствами и последовательными портами, которые обеспечивают прием/передачу символов и управление режимами работы устройств.

Кроме того, для печати текста в фоновом режиме в состав дистрибуции DOS входит программа PRINT.COM. Эта практически единственная "мультизадачная" утилита операционной системы является примером стандартной резидентной программы. Она позволяет выполнять параллельно с печатью текста другую работу (например, редактирование другого текста).

Специально следует отметить драйверы электронного диска и кэш-памяти. Драйвер электронного диска называется RAMDRIVE.SYS. Этот драйвер организует в расширенной или в дополнительной памяти компьютера быстрый псевдодиск. Можно организовать электронный диск и в основной памяти, но основной памяти всегда мало!

Операционная система MS-DOS 4.01 имеет в своем составе систему управления дополнительной и расширенной памятью, которая реализуется драйвером HIMEM.SYS. Этот драйвер позволяет программам, специальным образом составленным, использовать расширенную или дополнительную память для хранения данных (но не для выполнения программ).

С помощью драйвера SMARTDRV.SYS можно создать кэш-память для жесткого диска. Особенно эффективно применение кэш-памяти для работы с базами данных, когда вам периодически требуется одна и та же информация: в следующий раз, когда информация потребуется, она будет доступна без обращения к диску.

Такое широко распространенное и ставшее стандартным для любого персонального компьютера устройство, как мышь, не поддерживается операционной системой. Для использования мыши следует подключить драйвер, который обычно поставляется вместе с этим устройством.

Другие устройства ввода/вывода (сканеры, плоттеры, стриммеры и т. п.) также не поддерживаются DOS. Для этих устройств фирмы поставляют драйверы, которые надо подключить к операционной системе.

1.4. Обзор прерываний BIOS

Как правило, любая программа работает с тем или иным устройством ввода/вывода. Программы для первых ЭВМ работали непосредственно с портами и регистрами этих устройств. Модули, выполняющие такие стандартные действия, как ввод данных с перфокарт или печать результатов вычислений, входили в состав пользовательских программ.

В начале своего развития микропроцессорные системы имели в своем составе программу, называемую "монитор". Эта программа обычно находилась в постоянном запоминающем устройстве и обслуживала устройства ввода/вывода - клавиатуру, дисплей, кассетный накопитель на магнитной ленте и др. Диалоговая часть монитора позволяла выполнять некоторые операторские функции - загрузку и запуск программы, отладку программы в пошаговом режиме, печать текстов, просмотр и редактирование содержимого памяти и т. п. Но самое главное - прикладные программы, составленные для этих систем, могли пользоваться модулями монитора для работы с периферийной аппаратурой и выполнения других функций.

Программа уже не содержала все необходимые для ее работы модули, а пользовалась "стандартными" услугами программы-монитора. Такая организация программы не только уменьшила размер ее загрузочного модуля, но и позволила программистам сосредоточить свои усилия на решении основной задачи.

Как программа пользовалась услугами монитора? Механизм взаимодействия программы пользователя и монитора был реализован по-разному в разных системах. В худшем случае прикладная программа пользовалась известными абсолютными адресами модулей монитора, в лучшем - использовала специальные таблицы адресов программных модулей.

К сожалению, разные системы были несовместимы по составу модулей монитора и механизму их вызова, что сильно затрудняло, если не совершенно исключало их программную совместимость.

В первом массовом персональном компьютере IBM PC модули обслуживания стандартной периферии были записаны в постоянном запоминающем устройстве. Совокупность этих модулей (плюс программа начальной инициализации и тестирования) называется базовой системой ввода/вывода - Basic Input/Output System. общепринятое сокращение - BIOS.

Выпускаемые различными фирмами компьютеры, совместимые с IBM PC, могут немного отличаться по типу периферийного оборудования, но для достижения совместимости с IBM PC модули BIOS нивелируют эти различия, предоставляя в распоряжение программы пользователя стандартный набор модулей для работы с устройствами ввода/вывода.

Не будет преувеличением сказать, что одна из причин такого невиданного успеха компьютера IBM PC на рынке персональных компьютеров - наличие хорошо продуманного стандартного интерфейса модулей BIOS и прикладных программ. Именно благодаря этому интерфейсу достигается почти стопроцентная со-

вместимость компьютеров этого типа, выпускаемых разными фирмами, по программному обеспечению.

В этом разделе книги мы расскажем о том, как прикладные программы, составленные для компьютера, совместимого с IBM PC, могут пользоваться модулями BIOS для работы со стандартной периферией, затем приведем краткий обзор основных модулей BIOS.

Напомним вам, что такое программные прерывания, так как именно они используются для вызова модулей BIOS.

В начале оперативной памяти персонального компьютера (в пределах первого килобайта) находится так называемая векторная таблица прерываний. Она состоит из 256 ячеек, хранящих адреса программ-обработчиков прерывания. Мы будем подробно изучать эту таблицу в главе 4, а сейчас вспомним машинную команду `INT <n>`.

По этой команде содержимое ячейки векторной таблицы прерываний с номером `n` помещается в адресные регистры процессора, причем в стеке запоминается текущее содержимое адресных регистров. Управление передается по адресу, записанному в ячейке таблицы. Программа-обработчик прерывания должна заканчиваться командой `IRET`, по которой из стека извлекается старое значение адресных регистров и управление передается обратно в программу, вызвавшую прерывание командой `INT <n>`.

Вообще говоря, процедура вызова и обработки программного прерывания похожа на процедуру вызова подпрограммы, отличие заключается в том, что вызывающая программа "не знает" абсолютного адреса модуля обработки прерывания в памяти. Поэтому работа программ не зависит от адресов расположения модулей обработчиков прерывания.

Для вызова модуля BIOS программа использует команду `INT <n>` с соответствующим номером `n`. Программа передает параметры модулям BIOS через регистры процессора, результат работы модуля возвращается также в регистрах.

Не все номера прерываний `n` используются BIOS. Часть из них предназначена для аппаратных прерываний от устройств ввода/вывода, часть зарезервирована для DOS, часть - для программ пользователя.

Подробно прерывания BIOS будут изучаться во втором томе, так как описание этих прерываний трудно отделить от описания особенностей аппаратуры. Приводимый здесь обзор предназначен в основном для иллюстрации основных возможностей BIOS по обслуживанию периферии, поэтому в обзор вошли не все прерывания BIOS, а только самые важные. Остальные прерывания будут изучаться по мере необходимости при изложении соответствующего материала.

Получение информации об оборудовании

INT 11h - Получить список оборудования

Прежде чем пытаться работать с каким-либо устройством ввода/вывода, следует убедиться в том, что оно есть в составе оборудования компьютера. В процессе инициализации текстовые модули, находящиеся в BIOS, динамически оп-

ределяют состав аппаратного обеспечения машины и записывают конфигурацию системы в специально отведенную для этого ячейку памяти.

Программа, вызывая прерывание INT 11h, получает в регистре AX содержимое этой ячейки. Каждый бит в слове конфигурации отвечает за соответствующее устройство.

Анализируя слово конфигурации, программа может узнать, входят ли в состав оборудования компьютера дискеты и если входят, то сколько дисководов имеется в наличии, присутствует ли арифметический сопроцессор, какой начальный режим дисплейного адаптера используется, сколько в системе принтеров, адаптеров последовательного интерфейса RS-232, подключен ли игровой адаптер (джойстик).

Обычно прикладная программа не работает сама с аппаратурой, а пользуется услугами операционной системы. При обращении к стандартной аппаратуре через операционную систему программа пользователя получит признак ошибки, если запрашиваемое устройство отсутствует. Программы, составленные на языке программирования Си при использовании библиотеки эмуляции арифметического сопроцессора сами определяют, имеется сопроцессор или нет, и не пытаются пользоваться отсутствующим устройством. Операции сопроцессора эмулируются центральным процессором, и программа просто работает медленнее. Но, если программа обращается непосредственно к портам ввода/вывода отсутствующего устройства, это может привести в лучшем случае к зависанию системы.

INT 12h - Получить размер основной памяти

Сказанное выше справедливо и по отношению к оперативной памяти. Для работы некоторых программ требуется достаточное количество памяти. Прерывание INT 12h возвращает в регистре AX количество имеющихся блоков памяти размером в один килобайт. Анализируя эту величину, программы могут при нехватке памяти либо вывести на экран соответствующее сообщение и отказаться от работы, либо изменить алгоритмы работы, организовав, например, "виртуальную" память на диске или просто записывая в файл промежуточные результаты.

Если ваш компьютер оборудован расширенной памятью (адресное пространство этой памяти находится выше границы в 1 Мбайт), размер этой памяти в килобайтах можно узнать, вызвав прерывание INT 15h со значением регистра AX, равным 8800h.

Работа с дисплейным адаптером

Прерывание INT 10h выполняет все многочисленные операции по обслуживанию дисплейного адаптера.

При вызове прерывания INT 10h, как и при вызове многих других прерываний, регистр AH содержит номер функции, которую требуется выполнить. Остальные регистры при вызове прерывания содержат операнды.

Программирование дисплейного адаптера - сложная задача. Функции, выполняемые прерыванием INT 10h, обширны, полностью они будут описаны во втором томе книги. Приведем краткий обзор функций прерывания INT 10h.

00h - Задание видеорежима

Эта функция обычно вызывается первой при начале работы с дисплейным адаптером или при необходимости изменить текущий режим адаптера. Что здесь имеется в виду?

Напомним, что дисплейный адаптер может работать либо в текстовом, либо в графическом режиме. На самом деле существует несколько текстовых и несколько графических режимов, различающихся количеством строк и столбцов, способом представления цвета и т. д.

В процессе инициализации BIOS задает начальный режим адаптера, исходя из его типа. Если вашей программе нужен другой режим, отличный от исходного, она должна использовать эту функцию. При этом необходимо учитывать, что дисплейные адаптеры могут поддерживать не все режимы.

01h - Установить характеристики курсора

С помощью этой функции вы можете установить размер и форму курсора, сделать курсор мигающим или убрать его совсем.

02h - Установить положение курсора

Эта функция позволяет управлять расположением курсора на экране: в частности, один из способов убрать курсор - расположить его за пределами экрана, например на несуществующей 26 строке.

03h - Получить положение курсора

Программа может узнать в любое время, где расположен курсор. Это может потребоваться ей, например, для того, чтобы переместить курсор в следующую позицию (вправо, вверх, вниз, на 10 символов левее текущего положения и т. д.).

04h - Получить положение светового пера

Световое перо используется относительно редко, однако если оно есть, то функция 04h позволит вам работать с этим устройством.

05h - Выбрать активную страницу дисплейной памяти

Компьютер хранит, как правило, не один отображаемый образ экрана, а несколько. Для этого видеопамять (память для хранения видеоизображения, находится на плате видеоконтроллера) разбивается на так называемые страницы. Отображается только активная страница видеопамати. Используя механизм страниц, программа может подготовить заранее изображение в неактивной странице,

затем сделать подготовленную страницу активной. Изображение новой страницы мгновенно появится на экране.

Некоторые отладчики используют одну страницу видеопамати для отлаживаемой программы, другую - для выдачи своих диагностических сообщений.

06h, 07h - Прокрутка (скроллинг) окна вверх/вниз

С помощью этих функций можно переместить выбранную область окна на заданное число строк вверх или вниз. Освободившееся место будет заполняться пустыми строками. Одно из применений этой функции - полная очистка экрана.

08h, 09h - Прочитать/записать символ и атрибут

С помощью этих функций можно прочитать или записать символ и его атрибут. При записи символа можно задать число повторений. Функцию записи с повторением удобно использовать для заполнения области экрана каким-либо символом.

Запись/чтение символа начинается с текущего положения курсора.

0Ah - Записать символ

Эта функция предназначена для записи символа в видеопамать без задания для него индивидуального значения атрибута. Используется текущий атрибут. Можно задавать кратность записи.

0Bh - Выбрать цветовую палитру

Эта функция позволяет управлять цветом рамки вокруг изображения в текстовом режиме и цветом фона в графическом режиме (для адаптеров EGA, VGA).

0Ch, 0Dh - Записать/прочитать графическую точку

Можно высветить точку заданного цвета в заданном месте экрана для выбранной дисплейной страницы или соответственно прочитать значение цвета любой точки на экране.

0Eh - Запись символа на экран в стиле TTY

После записи символа на экран курсор продвигается на следующую позицию, при этом обрабатываются такие управляющие символы, как BEL (подача звукового сигнала), возврат на одну позицию, перевод строки, возврат к началу строки.

0Fh - Прочитать текущий видеорежим

Если ваша программа изменяет видеорежим, она может сохранить старый видеорежим, получив его с помощью этой функции.

Кроме того, функция возвращает количество столбцов на экране и номер текущей активной дисплейной страницы.

10h, 11h, 12h - Обслуживание адаптера EGA .. VGA

Эти функции работают только при использовании дисплейных адаптеров EGA и VGA. Они позволяют устанавливать свою цветовую палитру, загружать знакo-генератор (например, шрифтом для русских букв) и выполнять некоторые другие функции.

13h - Запись строки

Для машин класса AT и выше при наличии дисплейных адаптеров EGA или VGA эта функция позволяет вывести на экран произвольную строку символов заданной длины, с заданным атрибутом и в заданном месте экрана. Можно также задать номер дисплейной страницы.

Если вы не можете использовать эту функцию (ваш компьютер - XT или дисплейный адаптер - CGA), единственный способ вывести на экран строку символов с помощью прерывания INT 10h - вызывать в цикле функции 09h, 0Ah или 0Eh для вывода строки по одному символу.

Обслуживание клавиатуры

Обработчик прерывания INT 16h выполняет несколько функций, связанных с обслуживанием клавиатуры. Мы не будем сейчас перечислять эти функции, они будут подробно описаны в главе, посвященной клавиатуре. С помощью функций обслуживания клавиатуры можно выполнить ввод кода нажатой клавиши как с ожиданием нажатия, так и без ожидания - в последнем случае возвращается код последней нажатой клавиши.

Заметим, что символы, введенные с клавиатуры, помещаются в специальный клавиатурный буфер. Функция ввода символа без ожидания нажатия на клавишу проверяет состояние буфера - есть в нем символы, или нет. Если в буфере есть символы, последний помещенный в буфер символ возвращается программе. Этот символ затем может быть считан функцией ввода с ожиданием нажатия - фактически ожидания при этом не будет.

Для очистки буфера клавиатуры также можно использовать пару описанных выше функций - сначала программа проверяет, пуст ли буфер, и, если он не пуст, считывает символ. Считанный символ никуда не помещается (теряется). После считывания символа программа опять проверяет содержимое буфера и так до тех пор, пока клавиатурный буфер не окажется пустым.

Для машин класса не ниже AT обработчик прерывания INT 16h выполняет и другие функции - установку задержки, запись символов в буфер клавиатуры, обслуживание расширенной клавиатуры.

Обслуживание дисковой подсистемы

Прерывание INT 13h предназначено для обслуживания жестких и флоппи-дисков. Многочисленные функции прерывания INT 13h выполняют все операции по вводу/выводу на диски. Мы сделаем обзор только самых важных функций, остальные будут рассмотрены в книге 3, посвященной файловой системе.

00h - Сброс дисковой системы

Эта функция выполняет установку в исходное состояние всей дисковой системы или выбранного дискового устройства. Используется обычно перед началом работы с устройством.

01h - Получить состояние дисковой системы

Эта функция позволяет проверить результат выполнения предыдущей операции. Если операция завершилась аварийно, при помощи этой функции можно определить код ошибки.

02h/03h - Чтение/запись секторов

Выполняется чтение секторов в оперативную память компьютера или запись информации из памяти в секторы диска.

Сектор задается для выбранного устройства, дорожки и головки. Программа должна также задать количество читаемых/записываемых секторов.

04h - Проверка секторов

Функция проверяет секторы на правильность циклической контрольной суммы, CRC (Cyclic Redundancy Check); записи содержимого секторов в память не происходит.

Другие функции прерывания INT 13h

Среди других функций прерывания INT 13h - форматирование дорожки, тестирование и предустановка диска, запуск диагностики контроллера и многое другое. Описание этих функций мы отложим до глав, посвященных файловой системе.

Вывод на принтер (параллельный порт)

BIOS содержит простейшую поддержку принтера - три функции прерывания INT 17h. Это функция 01h - инициализация принтера, 02h - опрос состояния принтера и 00h - вывод символа на принтер.

Так как к персональному компьютеру можно подключить несколько последовательных портов, при обращении к принтеру следует указывать номер порта.

Обслуживание последовательного порта связи

Функции прерывания INT 14h обслуживают порт последовательной передачи данных RS-232. С помощью этих функций можно задавать формат и скорость передачи данных, определять состояние портов и, конечно, выполнять побайтную передачу данных.

Работа с системными часами

Функции прерывания INT 1Ah обслуживают часы, имеющиеся в каждом компьютере. С их помощью вы можете установить время и дату, опросить текущее состояние часов, вы можете работать с часами реального времени, которые имеются на машинах класса не ниже AT.

Для AT можно установить на заданное время "будильник" - в нужный момент будет вызвано прерывание "будильника" с номером 4Ah. Обработчик прерывания INT 4Ah может подать звуковой сигнал или вывести на экран предупреждающее сообщение.

Перезагрузка операционной системы

Вызов прикладной программой прерывания INT 19h приведет к перезагрузке операционной системы.

Расширенный сервис для машин класса AT

Прерывание INT 15h использовалось в компьютерах IBM PC и IBM PC Jr для управления кассетным накопителем на магнитной ленте (функции 0 - 3). Для машин класса AT и более высокого класса прерывание INT 15h имеет и другое назначение. С его помощью обслуживается расширенная клавиатура, выполняется программная задержка, задаваемая в микросекундах, обслуживается расширенная память. Кроме того, одна из функций прерывания INT 15h переводит процессор 80286 или 80386 в виртуальный (защищенный) режим. Заметим, что вернуть процессор обратно в реальный режим можно только сигналом начального сброса. Это же относится и к арифметическому сопроцессору 80287.

Функция C0h прерывания INT 15h выдает дополнительные сведения о конфигурации аппаратных средств компьютера.

Для PS/2 назначение некоторых функций этого прерывания другое по сравнению с машиной AT.

На этом мы завершим описание предоставляемых BIOS функций и перейдем к обзору функций DOS.

1.5. Обзор прерываний DOS

DOS предоставляет набор системных вызовов, реализованных с использованием механизма программных прерываний. Эти вызовы открывают прикладной программе доступ к системной информации, к системе консольного ввода/вывода, файловой системе, к подсистеме управления программами и памятью, позволяют организовать обращение к драйверам устройств ввода/вывода и т. д.

Все основные функции DOS вызываются с помощью прерывания INT 21h, однако DOS использует и другие прерывания:

- INT 20h - завершение работы программы
- INT 25h/26h - чтение/запись на диск с абсолютной адресацией секторов

INT 27h	- завершение работы программы с оставлением ее резидентной в памяти
INT 28h	- прерывание зарезервировано для DOS, может быть использовано для составления резидентных программ
INT 2Eh	- выполнение команды DOS
INT 2Fh	- прерывание мультиплексера, используется для спулера печати PRINT.COM.

Функции прерывания INT 21h можно разделить на следующие группы:

- получение системной информации
- символьный ввод/вывод
- работа с файловой системой
- управление программами
- управление памятью
- связь с драйверами устройств
- прочий системный сервис

В данном обзоре мы кратко рассмотрим эти группы, делая акцент в основном на составе функций. Полностью информация об использовании функций прерывания DOS INT 21h (и других прерываний DOS) будет приводиться в соответствующих разделах книги.

Номер функции задается при вызове прерывания INT 21h в регистре AH.

Получение системной информации

Функция 30h возвращает в регистре AX номер версии DOS. Например, для версии MS-DOS 4.00 содержимое регистра AH равно 00, регистра AL - 04. Дополнительно через регистр BH функция возвращает программе серийный номер фирмы-производителя OEM: 00 - IBM, 16h - DEC, 0FFh - Microsoft и т.п.; в регистрах BL:CH после вызова функции находится серийный номер пользователя. (= 0)

Эта информация может применяться для анализа возможности использования таких средств операционной системы, которые поддерживаются не всеми версиями DOS, или для настройки программы на конкретный серийный номер пользователя.

Функции 2Ah и 2Ch позволяют программе узнать системную дату и время.

Есть функции, возвращающие текущий диск и текущий каталог. Номера этих функций - 19h и 47h.

Функция 2Fh позволяет программе узнать адрес текущей области DTA (Disk Transfer Area), которая используется, например, при поиске файлов в каталоге.

Бажная информация находится в блоке PSP (Programm Segment Prefix). Этот блок располагается в памяти непосредственно перед выполняющейся программой. В нем находятся, в частности, параметры, передаваемые программе при запуске. Функция 62h возвращает адрес текущего блока PSP.

Кратко перечислим некоторые другие функции для получения системной информации:

35h - получить значение вектора прерывания с заданным номером

4Dh - узнать код завершения процесса

59h - получить расширенный код ошибки

54h - узнать, используется ли проверка при записи на диск

33h - узнать, используется ли проверка на CTRL+BREAK

Символьный ввод/вывод

Эти функции применяются для работы со всеми символьными устройствами, такими, как консоль, принтер, последовательный порт, и называются функциями стандартного ввода/вывода. Ввод/вывод программы, использующей стандартные функции, может быть переназначен.

Приведем обзор основных функций стандартного символьного ввода/вывода в виде таблицы:

<i>Код</i>	<i>Назначение</i>	<i>Описание</i>
01h	Ввод с клавиатуры	Выполняется ввод символа со стандартного ввода и эхо-вывод символа на стандартное устройство вывода. Выполняется проверка на нажатие комбинации клавиш CTRL+C и CTRL+BREAK
06h	Ввод с клавиатуры без ожидания	Ввод символа со стандартного ввода и вывод его на устройство стандартного вывода. Комбинации CTRL+C и CTRL+BREAK не проверяются
07h	Прямой ввод с клавиатуры	Ввод символа со стандартного устройства ввода. Комбинации клавиш CTRL+C и CTRL+BREAK не проверяются
08h	Ввод с клавиатуры	Аналогично функции 07h, но проверяются комбинации клавиш CTRL+C и CTRL+BREAK
02h	Отобразить символ	Отображаемый символ посылается на стандартное устройство вывода
09h	Отобразить строку символов	На стандартное устройство вывода посылается строка, закрытая символом '\$'
03h	Ввод из последовательного порта	Вводится символ из последовательного порта
04h	Вывод в последовательный порт	Выводится символ на последовательный порт
05h	Вывод на принтер	Выводится символ на принтер

Из таблицы видно, что для ввода с клавиатуры можно использовать несколько функций. Ввод без эхо-вывода удобен для такой информации, как пароли. Если логика работы программы не допускает прерывания по нажатию комбинаций клавиш CTRL+C или CTRL+BREAK, нужно использовать функции, которые не проверяют эти комбинации.

Для вывода строки символов можно использовать функцию 09h, но выводимая строка не может содержать символ '\$', т. к. этот символ используется в качестве признака конца строки.

Работа с файловой системой

DOS предоставляет программам обширный сервис для работы с файлами и дисками. Практически все файловые операции можно выполнять с помощью специально предназначенных для этого функций, только в некоторых случаях (защита данных от копирования, например) приходится использовать прямое обращение к диску.

С помощью файловых функций DOS можно создавать и удалять файлы и каталоги, открывать и закрывать файлы, создавать временные файлы. Ввод/вывод может быть буферизован, можно получить как последовательный доступ к содержимому файла, так и произвольный.

Мы не будем приводить сейчас конкретные примеры или номера функций, отложим это до книги 3, посвященной файловой системе.

Управление программами

DOS предоставляет программам возможность организовать запуск других программ или загрузку и выполнение программных оверлеев. Для этого служит функция 4Bh.

Для завершения работы программа должна также использовать одну из специальных функций DOS. Функция 4Ch, завершая работу программы, позволяет передать операционной системе некоторое число, называемое кодом завершения программы. Это число может быть затем проанализировано в пакетном файле командой IF ERRORLEVEL. Если одна программа запускает другую, то первая может получить код завершения второй с помощью функции 4Dh.

Для того чтобы завершающаяся программа осталась в оперативной памяти (т. е. стала резидентной), она должна вызвать прерывание INT 27h или воспользоваться функцией 31h.

Мы приведем различные примеры запуска программ из программ и научимся составлять резидентные программы.

Управление памятью

DOS управляет распределением памяти с помощью блоков управления памятью MCB (Memory Control Block). Вся память разбивается на блоки различного размера, которым предшествует блок MCB, содержащий характеристики данного блока памяти (например, его размер).

Программа может динамически получать и освобождать области памяти с помощью функций 48h и 49h соответственно. Можно также изменять размер блока, выделенного операционной системой программе. Это делает функция 4Ah.

Детально механизм управления памятью будет рассмотрен в главе 2 при описании векторной таблицы связи DOS.

Связь с драйверами устройств

Мы уже обращали ваше внимание на то, что программы не могут обращаться непосредственно к драйверам устройств ввода/вывода. Все обращения к драйверам имеют либо неявный характер (ввод/вывод с помощью функций прерывания INT 21h), либо используют специальную функцию DOS с кодом 44h. Эта функция используется для обмена управляющей информацией между драйвером и программой.

Другие функции

Существуют функции, позволяющие переопределять прерывания, работать с системными часами и календарем, функции для работы в сети и некоторые другие. Эти функции будут описаны позже в соответствующих разделах книги.

1.6. Обработка ошибок

Когда программа обращается к DOS для выполнения какой-либо операции, она должна вызвать соответствующее прерывание, загрузив перед вызовом прерывания все необходимые операнды в регистры процессора. Если выполнение операции невозможно по каким-то причинам (неправильные операнды, устройство неработоспособно, запрашиваемая операция не поддерживается текущей версией DOS и т. д.), то для большинства функций DOS устанавливается признак ошибки - флаг переноса CARRY. Для MS-DOS версии 2.0 и более поздних версий регистр AX при этом содержит код ошибки.

Приведем коды ошибок, возвращаемые программе через регистр AX:

- | | | |
|----|---|----------------------------------|
| 1 | - | Неправильный код функции |
| 2 | - | Файл не найден |
| 3 | - | Путь не найден |
| 4 | - | Слишком много открытых файлов |
| 5 | - | Доступ запрещен |
| 6 | - | Неправильный хендл |
| 7 | - | Разрушен блок управления памятью |
| 8 | - | Недостаточно памяти |
| 9 | - | Неправильный адрес блока памяти |
| 10 | - | Неправильная среда |
| 11 | - | Неправильный формат |
| 12 | - | Неправильный код доступа |
| 13 | - | Неправильные данные |

- 14 - Зарезервировано
- 15 - Ошибка при указании дисководов
- 16 - Невозможно удалить текущий каталог
- 17 - Другое устройство
- 18 - Больше нет подходящих файлов

Для MS-DOS версии 3.0 и более поздних версий обработка ошибок значительно расширена. Введена функция 59h прерывания INT 21h, предназначенная для получения дополнительной информации об ошибках.

При вызове этой функции регистр BX должен содержать индикатор уровня анализа ошибок, который должен быть равен 0. Кроме расширенного кода ошибки, возвращаемого в регистре AX, программа может получить класс ошибки (регистр BH), код предполагаемых действий (регистр BL), локализацию ошибки, т. е. место, где произошла ошибка (регистр CH).

К сожалению, эта функция разрушает содержимое регистров CL, DX, SI, DI, BP, DS, ES. Программа, использующая функцию 59h, должна позаботиться о сохранении содержимого этих регистров.

Расширенный код ошибки, возвращаемый в регистре AX, может принимать значения, указанные в приводимой ниже таблице. Коды от 1 до 18 эквивалентны представленным выше и второй раз не приводятся.

Код	Значение
19	Запись на защищенный от записи диск
20	Задан неизвестный идентификатор устройства
21	Дисковод не готов
22	Неизвестная команда
23	Ошибка циклического кода проверки
24	Неправильная длина структуры запроса
25	Ошибка поиска
26	Неизвестен тип среды носителя данных
27	Сектор не найден
28	Кончилась бумага в принтере
29	Ошибка записи
30	Ошибка чтения
31	Общая ошибка
32	Нарушение разделения файла
33	Нарушение блокировки файла
34	Неправильная замена диска
35	FCB недоступен (слишком много блоков FCB)
36	Переполнился буфер разделения
37	Зарезервировано
38	Не завершена операция "Конец файла"
39-49	Зарезервировано

50	Сетевая функция не поддерживается
51	Удаленный компьютер "не слышит"
52	Дублирование имени в сети
53	Сетевое имя не найдено
54	Сеть занята
55	Сетевое устройство больше не существует
56	Превышен лимит команды сетевой BIOS
57	Ошибка в аппаратуре сетевого адаптера
58	Неправильный ответ из сети
59	Непредусмотренная ошибка сети
60	Несовместимый удаленный адаптер
61	Заполнена очередь печати
62	Для печатаемого файла недостаточно места
63	Печатающийся файл был удален
64	Сетевое имя было удалено
65	Доступ запрещен
66	Неправильный тип сетевого устройства
67	Сетевое имя не найдено
68	Превышен лимит сетевого имени
69	Превышен лимит сеанса сетевой BIOS
70	Временная пауза
71	Сетевой запрос отвергнут
72	Приостановлена печать или переадресация диска
73-79	Зарезервировано
80	Файл уже существует
81	Зарезервировано
82	Невозможно создать дескриптор в каталоге
83	Ошибка обработчика критических ошибок INT 24h
84	Слишком много переназначений
85	Двойное переназначение
86	Неправильный пароль
87	Неправильный параметр
88	Ошибка данных в сети
89	Нет такой функции в сети
90	Требуемый компонент системы не установлен

Класс ошибки, передаваемый в регистре ВН, содержит информацию, которая поможет вам обработать данную ошибку:

- | | |
|---|---|
| 1 | - Недостаточно ресурсов: блоков FCB, памяти и т. д. |
| 2 | - Временная ситуация |

-
- | | |
|----|---|
| 3 | - Нет прав доступа |
| 4 | - Внутренняя ошибка DOS |
| 5 | - Ошибка аппаратуры |
| 6 | - Системная ошибка DOS (нет CONFIG.SYS и т. п.) |
| 7 | - Ошибка в прикладной программе |
| 8 | - Файл или объект не найден |
| 9 | - Неправильный формат файла или объекта |
| 10 | - Файл или объект заблокирован |
| 11 | - Ошибка носителя данных |
| 12 | - Файл или объект уже существует |
| 13 | - Прочие ошибки |

Код предполагаемых действий, передаваемый через регистр BL, поможет вашей программе правильно обработать ошибку и избежать зависания системы. Приведем таблицу кодов предполагаемых действий:

<i>Код</i>	<i>Действие</i>
1	Повторить операцию позже. Можно спросить пользователя, желает он повторить операцию или завершить работу программы
2	Повторить предыдущую операцию после небольшой паузы. Если ошибка не исчезла, следует спросить пользователя, будет он ждать и дальше или следует завершить работу программы
3	Если пользователь вводил какие-то данные для DOS, следует попросить его ввести эти данные еще раз (например, пользователь мог указать неправильный идентификатор диска или путь доступа к файлу)
4	Аварийно завершить работу прикладной программы с выполнением всех обычных завершающих действий (закрытие файлов, сброс буферов на диск, освобождение блоков памяти и т. д.)
5	Немедленный выход из программы без выполнения завершающих действий. Система находится в непредсказуемом состоянии
6	Следует игнорировать ошибку
7	Повторить операцию после того, как пользователь выполнит требуемые действия (установит дискету и т. п.)

Сведения о локализации ошибки передаются в регистре CH.

Приведем таблицу кодов локализации:

<i>Код</i>	<i>Локализация ошибки</i>
1	Локализация ошибки не может быть определена (система не знает, где произошла ошибка)
2	Ошибка произошла в блочном устройстве (диск или магнитная лента)
3	Ошибка связана с сетью
4	Ошибка произошла в символьном устройстве, например в принтере
5	Ошибка связана с оперативной памятью

Если ваша программа составлена на языке Ассемблера, то после обращения к DOS через прерывание следует проверить состояние флага переноса:

```
int 21h
jc  error
```

Программы, составленные на языке Си, обращаются к прерываниям DOS обычно с помощью таких функций, как `intdos`, `int86`, `intdosx` и т. д. Для передачи параметров используются структуры `REGS`, `WORDREGS`, `BYTEREGS`, `SREGS`. Они описаны в файле `dos.h`; для использования этих структур программа должна содержать строку:

```
include <dos.h>
```

Значение флага переноса записывается в переменную `cflag`, определенную в структуре `WORDREGS`. Эта структура входит в объединение `REGS`:

```
union REGS {
    struct WORDREGS x;
    struct BYTEREGS h;
}
```

```
struct WORDREGS {
    unsigned int ax;
    unsigned int bx;
    unsigned int cx;
    unsigned int dx;
    unsigned int si;
    unsigned int di;
    unsigned int cflag;
}
```

```
struct BYTEREGS {
    unsigned char al, ah;
    unsigned char bl, bh;
    unsigned char cl, ch;
    unsigned char dl, dh;
}
```


Проверка переменной cflag может быть выполнена, например, таким образом:

```
union REGS inregs, outregs;

intdos(&inregs, &outregs);
if( outregs.x.cflag != 0 ) error();
```

Код ошибки при этом содержится в переменной outregs.x.ax.

Приведем пример программы, которая стирает каталог с именем DIR в текущем каталоге и в случае ошибки выводит расширенную информацию об ошибке, класс ошибки, код предполагаемых действий и код локализации ошибки:

```
#include <dos.h>
#include <stdio.h>

union REGS inregs, outregs;
struct SREGS segregs;

void main(void);

void main(void) {
    char _far *dir_name = "DIR";

    // Стираем каталог с именем DIR. Для этого вызываем
    // функцию 0x3A прерывания INT 21h.

    inregs.h.ah = 0x3a;
    segregs.ds = FP_SEG(dir_name);
    inregs.x.dx = FP_OFF(dir_name);
    intdosx(&inregs, &outregs, &segregs);

    // Если после выполнения прерывания установлен
    // флаг переноса, выводим сообщение об ошибке.

    if(outregs.x.cflag != 0) {
        printf( "Ошибка при удалении каталога: %d",
            outregs.x.ax);
    }

    // Получаем расширенную информацию об ошибке
    // с помощью функции 0x59 прерывания INT 21h.

    inregs.h.ah = 0x59;
    inregs.x.bx = 0;

    // Сохраняем регистры в стеке, т. к. их содержимое изменится

    _asm {
        push ds
        push es
        push si
        push di
    }
```



```
intdosx(&inregs, &outregs, &segregs);
```

```
_asm {
    pop di
    pop si
    pop es
    pop ds
}
```

```
// Выводим расширенную информацию об ошибке
```

```
printf("\nРасширенный код ошибки:      %d"
       "\nКласс ошибки:                %d"
       "\nПредполагаемые действия:      %d"
       "\nЛокализация ошибки:          %d",
       outregs.x.ax,
       outregs.h.bh,
       outregs.h.bl,
       outregs.h.ch);
```

```
}
```

При составлении программ обработки ошибок следует учитывать, что для MS-DOS версии 1.0 при некоторых ошибках функции DOS возвращают в регистре AX значение 0FFh. Начиная с версии MS-DOS 2.0 при ошибке устанавливается флаг переноса, код ошибки записывается в регистр AX. Однако для более полной диагностики причины ошибки следует использовать функцию 59h прерывания INT 21h.

Если ваша программа, составленная на языке Си, вызывает функции DOS неявным образом (через функции стандартной библиотеки транслятора, такие, как `fprintf`, `puts` и т. д.), то можно воспользоваться средствами обработки ошибок, входящими в состав стандартной библиотеки.

Когда при обращении к функциям DOS средствами стандартной библиотеки транслятора Си возникает ошибка, то в глобальную переменную `errno` записывается код ошибки.

Возможны следующие коды ошибок (они описаны в файле `errno.h` и `stdlib.h`):

EINVAL Нет порожденных процессов.

Задача, не имеющая подзадач, выдала команду ожидания, или была выдана команда ожидания для подзадачи, имеющей признак NO-WAIT.

EAGAIN Больше нет процессов.

Попытка создать новый процесс окончилась неудачно, т. к. либо больше нет резервов для создания процессов, либо недостаточно оперативной памяти, либо превышен максимальный уровень вложенности процессов.

ENOMEM Слишком велик список аргументов.

Либо размер списка аргументов превышает 128 байт, либо требуемый размер памяти для среды превышает 32 Кбайта.

EACCESS	Доступ запрещен. Затребованный вид доступа к файлу запрещен или несовместим с установленными атрибутами файла (или каталога). Этот код ошибки передается при попытке чтения из неоткрытого файла, при попытке записи в файл, защищенный от записи, или при попытке открыть каталог как файл.
EBADF	Плохой номер файла (хендл). Хендл, использованный при вызове функции, имеет неверное значение или не относится к открытому файлу, или сделана попытка записи в открытый только для чтения файл или устройство.
EDEADLOCK	Произошла блокировка ресурсов. Произведено 10 неудачных попыток заблокировать файл. Этот код ошибки используется только MS-DOS версии 3.0 и более поздних версий.
EDOM	Ошибка в аргументе математической функции. Аргумент математической функции вышел за пределы области определения этой функции.
EEXIST	Файл уже существует. Сделана попытка создать файл с именем, которое уже используется существующим файлом.
EINVAL	Неверный аргумент. Для одного из аргументов функции было задано неверное значение.
EMFILE	Открыто слишком много файлов. Исчерпан запас номеров файлов (хендлов), больше нельзя открыть ни одного файла.
ENOENT	Нет такого файла или каталога. Запрошенный файл или каталог отсутствует или не может быть найден.
ENOEXEC	Ошибка формата загрузочного файла. Сделана попытка выполнить загрузочный файл, имеющий неправильный формат.
ENOMEM	Недостаточно памяти. Эта ошибка появляется, когда недостаточно памяти для запуска процесса или для удовлетворения запроса программы на выделение блока памяти.
ENOSPC	Нет свободного места на устройстве. На устройстве нет места для записи информации (например, переполнился диск).

ERANGE	Слишком большой результат. Слишком большой по величине аргумент математической функции привел к частичной или полной потере значимости результата.
EXDEV	Связь различных устройств. Сделана попытка переслать файл на другое устройство, используя функцию переименования.

Из приведенного выше списка кодов ошибок видно, что средствами стандартной библиотеки транслятора обрабатываются не только ошибки, возникающие при обращении к функциям DOS, но и ошибки, появляющиеся при работе с математическими функциями.

Для диагностической выдачи сообщения об ошибке можно использовать функции `perror` и `strerror`. Первая функция выводит в `stderr` соответствующее сообщение об ошибке, вторая только формирует строку сообщения. Функции `perror` и `strerror` имеют операнд - указатель на строку. Эта строка добавляется в начало стандартного сообщения об ошибке. Если к стандартному сообщению ничего добавлять не надо, операнд должен иметь значение `NULL`.

Следует заметить, что значение переменной `errno` отражает последнюю ошибку. Успешный вызов функции не приводит к автоматическому сбросу переменной `errno`.

Поэтому функции `perror` и `strerror` необходимо вызывать сразу после того, как вызываемая функция возвратит признак ошибки.

Приведем пример программы, обрабатывающей ошибки с использованием переменной `errno`:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <errno.h>

void main(int argc, char *argv[]) {
    FILE *stream;

    // Открываем файл только для чтения
    stream = fopen(argv[1], "r");

    // Если произошла ошибка, выводим сообщение
    if( (stream == NULL) || (ferror(stream)) ) {
        perror("Не могу открыть файл");
        exit(errno);
    }

    // Пытаемся произвести запись в файл, который
    // открыт только для чтения. Это приведет к ошибке.
```



```
fprintf(stream, "Пишем в файл\n");  
if( (stream == NULL) || (ferror(stream)) ) {  
// Выводим сообщение об ошибке двумя способами -  
// с помощью функций perror и strerror  
  
    perror("Запись в защищенный файл");  
    printf("Запись в защищенный файл: %s\n",  
        strerror(errno));  
  
    exit(errno);  
}  
exit( 0 );  
}
```

DOS имеет еще одно средство для обработки ошибок - обработчик критических ошибок (Critical Error Handler). Этот модуль вызывается DOS, когда она получает сообщение об ошибке от драйвера устройства.

Модуль выдает на экран хорошо известное вам сообщение:

Abort, Retry, Ignore?

Это сообщение обычно появляется, когда вы забываете вставить дискету или начинаете печатать при отключенном принтере.

Прикладные программы могут подключать свой модуль обработки критических ошибок вместо стандартного. Мы научимся обрабатывать критические ошибки в книге 3. Там же будет приведен соответствующий пример.

Уважаемые читатели!

АО "ДИАЛОГ-МИФИ" выпускает литературу по программированию и вычислительной технике, рассчитанную на широкий круг пользователей персональных компьютеров. В настоящее время имеются в продаже и в ближайшее время выйдут из печати следующие издания:

**ОПЕРАЦИОННАЯ СИСТЕМА MS-DOS, в 2 частях: 1 и 2 кн., 3 кн.
("Библиотека системного программиста")**

А. В. Фролов, Г. В. Фролов

Учебно-справочное руководство. От аналогичных изданий отличается более глубоким изложением материала. Описываются детали, которые часто остаются скрытыми даже для опытного программиста: внутренняя структура и организация работы MS-DOS, недокументированные прерывания. Разбираются профессиональные приемы работы. Приводятся исходные тексты программ с комментариями.

Объем каждой части 10 л., цена - 30 руб.

Продолжение на стр. 68, 98, 220, 236

ВЕКТОРНАЯ ТАБЛИЦА СВЯЗИ DOS

Вслед за областью данных BIOS в оперативной памяти IBM PC располагается область данных DOS. Здесь располагаются внутренние переменные и структуры DOS. Основные структуры данных организованы в виде дерева. Корнем является векторная таблица связи, которая содержит адреса всех остальных структур - список блоков управления памятью (MSB), список блоков управления устройствами DOS, таблица файлов, дисковые буфера.

Векторная таблица связи содержит и другую полезную информацию, открывающую доступ практически ко всем внутренним структурам данных операционной системы. Можно, например, получить доступ ко всем резидентным и загружаемым драйверам операционной системы. Можно узнать, какие дисковые устройства установлены в системе и какие у них характеристики. Зная форматы управляющих блоков операционной системы, можно анализировать ошибочные ситуации, возникающие при отладке программного обеспечения, разрабатывать программы, отображающие внутреннее состояние системы и конфигурацию устройств.

Понимание внутренней структуры MS-DOS едва ли не самое важное для профессионального системного программирования, поэтому это первое, на чем мы подробно остановимся. В последующих главах мы будем постоянно пользоваться этой информацией.

Многочисленные управляющие блоки, которые использует файловая система и BIOS при работе с дисковыми устройствами, будут описаны в книге 3, посвященной файловой системе.

Прикладная программа может пользоваться услугами DOS, вызывая прерывание INT 21H. Это прерывание имеет несколько десятков функций, которые и представляют собой интерфейс между DOS и прикладной программой. С помощью этих функций прикладная программа получает доступ к файловой системе, может обращаться к драйверам устройств, получать и устанавливать системные параметры, работать с дисплеем и клавиатурой и т. д.

Для полного использования возможностей прерываний DOS необходимы сведения о форматах различных управляющих блоков, используемых этими прерываниями.

2.1. Таблица связи управляющих блоков MS-DOS

Операционная система MS-DOS подобно операционным системам для больших ЭВМ серии ЕС содержит векторную таблицу связи основных управляющих блоков. К сожалению, в руководстве по MS-DOS ничего не говорится об этой таблице. Мы попытаемся в некоторой степени восполнить этот пробел, т. к. изучение векторной таблицы связи позволит глубже осознать принципы работы

операционной системы. Информация из векторной таблицы связи будет полезна для составления программ отображения распределения памяти, вывода списка загруженных драйверов, списка устройств прямого доступа и т. д.

Для получения адреса векторной таблицы связи можно воспользоваться недокументированной внутренней функцией 52h прерывания 21h. Для версий MS-DOS 2.x, 3.x, 4.00 и 4.01 после вызова этой функции регистры ES:BX будут содержать искомый адрес. Т. к. описание этой функции отсутствует в руководстве по MS-DOS, в следующих версиях операционной системы, возможно, придется искать другой способ получения адреса векторной таблицы связи. Может также измениться формат этой таблицы.

Функции для получения адреса векторной таблицы связи:

```

; **
; .Name      get_cvt
;
; .Title     Получить адрес векторной таблицы связи
;
; .Descr     Функция возвращает адрес векторной таблицы связи
;             в регистрах ES:BX для DOS версий 2.x, 3.x, 4.00, 4.01
;
; .Params    Нет
;
; .Return    ES - сегмент векторной таблицы связи,
;             BX - смещение векторной таблицы связи
; **
        PUBLIC get_cvt
        .MODEL tiny

        .CODE
get_cvt proc near

        mov     ax, 5200h
        int     21h
        ret

get_cvt endp
end

```

```

/ **
* .Name      get_cvt
*
* .Title     Получить адрес векторной таблицы связи
*
* .Descr     Функция возвращает адрес векторной таблицы связи
*             для DOS версий 2.x, 3.x, 4.00, 4.01
*
* .Params    Нет
*
* .Return    Указатель на векторную таблицу связи
* **/

```



```

#include <dos.h>
#include <stdio.h>
#include "sysp.h"

void far *get_cvt(void)
{
    union REGS inregs, outregs;
    struct SREGS segregs;

    inregs.h.ah = 0x52;
    intdosx( &inregs, &outregs, &segregs );

    return(FP_MAKE(segregs.es, outregs.x.bx));
}

```

В примере на языке Ассемблера процедура `get_cvt` вызывает функцию `52h` прерывания `21h`, после чего возвращает адрес векторной таблицы связи в регистрах `ES:BX`.

Функция `get_cvt`, составленная на языке Си, также вызывает это прерывание, пользуясь функцией `intdosx` стандартной библиотеки транслятора. Функция `intdosx` вызывает прерывание `21h`. В качестве параметров ей передаются адреса структур и объединения `inregs`, `outregs` и `segregs`. Объединение `inregs` должно содержать значения регистров перед вызовом прерывания, в объединение `outregs` и структуру `segregs` заносятся значения регистров общего назначения и сегментных регистров после того, как завершится обработка прерывания.

Подробнее об этом сказано в описании стандартной библиотеки.

Макро `FP_MAKE` описано в файле `sysp.h` и предназначено для конструирования FAR-указателя из значений сегмента и смещения:

```

#define FP_MAKE(seg, off) ((void far *) \
    (((unsigned long) (unsigned)(seg)) << 16L) | \
    ((unsigned long) (unsigned) (off))))

```

Следующая программа, написанная на языке Си, выводит на консоль адрес векторной таблицы связи:

```

#include <stdio.h>
#include "sysp.h"

void main(void)
{
    void far *cvt;
    cvt=get_cvt();
    printf("Адрес векторной таблицы связи: %Fp\n", cvt);
    exit(0);
}

```

Спецификация формата вывода `%Fp` используется для вывода значения указателя типа `FAR`.

Описание векторной таблицы связи, а также других управляющих блоков будет состоять из строк следующего вида:

(+4) 4 `filename_ptr` - указатель на имя файла

В этой строке:

- (+4) - смещение поля в управляющем поле в байтах;
- 4 - длина поля в байтах;
- filename - имя поля.

Вслед за именем идет краткое описание содержимого поля.

Итак, векторная таблица связи MS-DOS:

- (-2) 2 mcb_seg - сегмент первого управляющего блока памяти (MCB)
- (0) 4 dev_ch - указатель на первый блок управления устройствами DOS (DOS Device Control Block)
- (+4) 4 file_tab - указатель на таблицу файлов DOS
- (+8) 4 clock_dr - указатель на драйвер CLOCK\$, установленный или резидентный
- (+12) 4 con_dr - указатель на актуальный драйвер CON, установленный или резидентный

----- DOS 2.x -----

- (+16) 1 num_lgdr - число логических драйверов в системе
- (+17) 2 max_btbl - максимальное число М/блоков любого блочного устройства
- (+19) 4 disk_buf - указатель на первый дисковый буфер
- (+23) - null_dr - начало драйвера NUL - первого драйвера в списке драйверов DOS

----- DOS 3.x, 4.x -----

- (+16) 2 max_btbl - максимальное число байт в блоке блочного устройства
- (+18) 4 disk_buf - указатель на первый дисковый буфер
- (+22) 4 drv_info - указатель на массив информации об устройствах
- (+26) 4 fcb_tabl - указатель на таблицу FCB
- (+30) 2 fcb_size - размер таблицы FCB
- (+32) 1 num_bdev - число блочных устройств
- (+33) 1 lastdriv - значение LASTDRIVE в файле CONFIG.SYS (по умолчанию равно 5)
- (+34) - null_dr - начало драйвера NUL - первого драйвера в списке драйверов DOS

Функция `get_cvt` возвращает адрес поля `dev_cb`. Для удобства работы с векторной таблицей связи определим тип `CVT`:

```
#pragma pack(1)

typedef struct _CVT_ {
    unsigned mcb_seg;
    void far *dev_cb;
    void far *file_tab;
    void far *clock_dr;
    void far *con_dr;
    unsigned max_btbl;
    void far *disk_buf;
    void far *drv_info;
    void far *fcb_tabl;
    unsigned fcb_size;
    unsigned char num_bdev;
    unsigned char lastdriv;
} CVT;
```

```
#pragma pack()
```

Эта структура содержит описание полей векторной таблицы связи для MS-DOS версий 3.x и 4.x.

Директива `#pragma pack(1)` предназначена для выравнивания полей структуры на границу байта. Эта директива необходима потому, что по умолчанию транслятор Microsoft выравнивает поля в структуре на границу 16 битового слова. Неправильное выравнивание может привести к тому, что поля структуры не будут располагаться в памяти последовательно.

Заметьте, что функция `get_cvt` возвращает указатель на поле `dev_cb`. Модифицируем эту функцию так, чтобы можно было использовать для обращения к полям векторной таблицы связи структуру `_CVT_`:

```
/**
 * .Name      get_mcv
 *
 * .Title     Получить адрес векторной таблицы связи
 *
 * .Descr     Функция возвращает адрес векторной таблицы связи
 *            для DOS версий 2.x, 3.x, 4.00, 4.01
 *
 * .Params    Нет
 *
 * .Return    Указатель на векторную таблицу связи
 */
```

```
#include <dos.h>
#include <stdio.h>
#include "sysp.h"
```

```
CVT far *get_mcv(void)
{
    union REGS inregs, outregs;
```



```

struct SREGS segregs;
inregs.h.ah = 0x52;
intdosx( &inregs, &outregs, &segregs );
return((CVT far*)FP_MAKE(segregs.es,outregs.x.bx-2));
}

```

Ниже будут подробно описаны отдельные поля векторной таблицы связи и приведены примеры использования информации из этой таблицы.

2.2. Блоки управления памятью в MS-DOS

Первое поле векторной таблицы связи `mcb_seg` содержит сегментный адрес первого блока управления памятью (MCB). Блок MCB всегда начинается на границе параграфа, поэтому полный адрес первого блока будет равен `mcb_seg:0`.

Для лучшего понимания механизма управления памятью в MS-DOS вспомним карту распределения памяти:

Диапазон адресов	Содержимое
0000:0000	Векторы прерываний
0000:0400	Область данных BIOS
0000:0500	Область данных DOS
xxxx:0000	Область программ DOS (расширение BIOS, обработчики прерываний DOS, буфера, области данных, загружаемые драйверы устройств)
xxxx:0000	Резидентная порция COMMAND.COM
xxxx:0000	TSR-программы (остающиеся резидентными после запуска)
xxxx:0000	Выполняющиеся прикладные программы типа COM или EXE
xxxx:0000	Транзитная порция COMMAND.COM
A000:0000	Память EGA, используемая некоторыми видеорежимами
B000:0000	Память монохромного дисплейного адаптера
B800:0000	Память видеоадаптера CGA
C800:0000	Внешнее ПЗУ
F600:0000	ПЗУ интерпретатора BASIC
FE00:0000	ПЗУ BIOS

Первый килобайт памяти занимает таблица векторов прерываний. Она содержит 256 4 байтных элементов - дальние адреса обработчиков прерываний. Подробно формат и использование этой таблицы будет обсуждаться в главе, посвященной прерываниям.

Адреса 0000:0400 - 0000:04FF (или от 0040:0000 до 0050:0000) занимают область данных BIOS. Это внутренние переменные BIOS. К ним можно обращаться для получения различной информации, но необходимо только помнить, что формат этой области может быть различным для различных версий BIOS. Мы будем при необходимости ссылаться на отдельные поля этой области.

Начиная с адреса 0000:0500 (или с адреса 0050:0000, что одно и то же) следует область данных DOS. Здесь DOS хранит свои внутренние таблицы и переменные. Формат этой области (и ее размер) зависит от версии операционной системы.

Далее следует большая область памяти, используемая DOS, в которой располагаются:

- система ввода/вывода DOS (содержимое файла IO.SYS)
- обработчики прерываний DOS, в частности обработчик прерывания INT 21H (эти обработчики входят в состав файла MSDOS.SYS)
- внутренние буфера DOS и области данных
- загружаемые драйверы (описанные в файле CONFIG.SYS)

После драйверов располагается резидентная порция COMMAND.COM (командный интерпретатор). Она, в частности, обрабатывает прерывания INT 22H, INT 23H и INT 24H.

Следующая область памяти занимается программами, остающимися резидентными после запуска (TSR-программы).

После резидентных программ находится выполняющаяся в настоящий момент программа (COM или EXE). Она может занимать всю оставшуюся память до адреса A000:0000 или только часть этой памяти (для EXE-программ). Для EXE-программ можно при редактировании указать требуемый объем памяти.

Нижнюю часть 640-килобайтного адресного пространства (до адреса A000:0000) занимает транзитная часть COMMAND.COM. Она может перекрываться выполняющейся программой. Если программа перекроет транзитную часть COMMAND.COM, то после завершения выполнения программы эта часть командного интерпретатора будет загружена заново.

Область адресов от A000:0000 до C800:0000 используется дисплейными адаптерами. Каждый адаптер использует эту часть памяти по-своему.

Далее и до конца мегабайтной границы идет область ПЗУ. Там расположено ПЗУ BIOS, ПЗУ интерпретатора BASIC, расширение BIOS (например, EGA BIOS для дисплейных адаптеров EGA).

Диапазон адресов свыше мегабайта используется для машин класса не ниже AT. Это так называемая расширенная память (Extended Memory). Она обслуживается BIOS и используется операционной системой MS-DOS для организации

"электронного" диска, кэш-памяти для дисков. Некоторые прикладные программы (например, отладчик Microsoft CodeView) хранят в этой области свои данные. Полностью управлять расширенной памятью способны операционные системы типа OS/2 и UNIX в защищенном режиме работы (процессоры Intel 80286, 80386, 80486).

С расширенной памятью не надо путать дополнительную память (Expanded Memory). Эта память с помощью специальной аппаратуры и драйверов отображается в область адресного пространства, лежащую до границы 1 Мбайт. MS-DOS может использовать эту память аналогично расширенной памяти.

Зона памяти начиная с области программ DOS и до видеопамати дисплейных адаптеров разбита на блоки. Перед каждым блоком находится блок управления памятью - Memory Control Block (MCB).

Сегментный адрес первого блока MCB находится в векторной таблице связи в поле `mcv_seg` (смещение блока равно 0). Внутри блока MCB содержится длина описываемого данным MCB блока памяти. Следующий MCB начинается сразу за предыдущим. Таким образом, все блоки управления памятью связаны в список.

Блоки MCB бывают двух типов - M и Z. M-блоки ('middle') - это промежуточные блоки. Блок типа Z является последним блоком в списке и может быть только один.

Приведем формат блока MCB (описание этого блока отсутствует в документации по MS-DOS, поэтому его формат может измениться в последующих версиях операционной системы):

(0)	1	type	- тип блока MCB (M или Z)
(+1)	2	owner	- параграф владельца блока (если 0, то блок описывает сам себя)
(+3)	2	size	- число параграфов в этом блоке (один параграф имеет размер 16 байт)
(+5)	11	reserve	- зарезервировано

Для удобства работы файл `sysr.h` содержит определение типа MCB:

```
#pragma pack(1)
```

```
typedef struct _MCB_ {  
    unsigned char type;  
    unsigned owner;  
    unsigned size;  
    char reserve[11];  
} MCB;
```

```
#pragma pack()
```

Существует несколько удобных программ для просмотра списка блоков MCB.

Вот какую информацию выдает программа MI.COM из пакета PCSHELL при запуске с параметром /A:

Memory Info v5.8

Copyright 1989 Central Point Software, Inc. All rights reserved.

Conventional memory. Total: 640k

Largest executable program: 485k

Type	Paragraphs	Bytes	Owner
Sys	0BA4-18C5h	53792	0008h < DOS >
Free	18C7-18CFh	144	0000h < DOS >
Env	18D1-18D2h	32	18D4h JYRKEYB
Prog	18D4-1904h	784	18D4h JYRKEYB C:\DOS\JYRKEYB.COM C
Prog	1906-1A69h	5696	1906h COMMAND
Env	1A6B-1A7Dh	304	1906h COMMAND
	1A7F-1A82h	64	1906h COMMAND
Free	1A84-1A93h	256	0000h < DOS >
Prog	1A95-1DD8h	13376	1A95h MOUSE
Env	1DDA-1DEDh	320	1ED8h NS
Prog	1DEF-1ED6h	3712	1DEFh SHELLB DOSSHELL
Prog	1ED8-21EBh	12608	1ED8h NS f:\norton\NS.EXE
Env	21ED-2200h	320	2202h NC
Prog	2202-2527h	12896	2202h NC f:\norton\NC.EXESocha
	2529-253Ch	320	253Eh COMMAND
Prog	253E-26A1h	5696	253Eh COMMAND /a
Env	26A3-26B5h	304	253Eh COMMAND
Env	26B7-26CAh	320	26CCh MI
Prog	26CC-9FFFh	485k	26CCh MI c:\dos\MI.COM /a

Программа сообщает размер оперативной памяти (640 Кбайт), максимальный размер выполняемой программы (485 Кбайт), затем выдает список блоков памяти с указанием типа, занимаемых параграфов памяти, размера в байтах и владельца блока памяти. Программа MI различает четыре типа блоков памяти:

- системный (Sys), его владельцем является MS-DOS
- свободный (Free), обычно тоже принадлежит MS-DOS
- программный (Prog), его занимает запущенная программа
- среда (Env), содержит переменные среды MS-DOS

Откуда MI берет информацию о типе блока памяти и имени программы? Системный блок распознается по занимаемым адресам, программный - по наличию правильного префикса программного сегмента (будет описан ниже). Блок переменных среды находится перед программным блоком и содержит кроме собственно переменных среды еще и полный путь файла запущенной программы. Если блок не системный, программный или не является блоком среды и если в поле владельца этого блока записан ноль, программа отмечает такой блок памяти как свободный.

Видно, что для каждой запущенной программы создается два блока памяти - блок среды и программный блок. Среда формируется при загрузке операционной

системы с помощью команд SET и содержит строки вида:

```
LIB=D:\C600\LIB
```

Строки хранятся в формате ASCIIZ, т. е. закрыты двоичным нулем. Вся таблица переменных среды также закрыта двоичным нулем. После переменных среды в блоке памяти, отведенном для среды, содержится путь программного файла в формате ASCIIZ.

Блок памяти типа Prog (программный) независимо от формата загрузочного модуля (COM или EXE) начинается с префикса программного сегмента PSP, за которым следует сама программа.

Таким образом, при загрузке для программы выделяются блоки памяти, располагающиеся в следующей последовательности:

- МСВ для блока памяти переменных среды
- блок памяти переменных среды
- МСВ программного блока памяти
- префикс программного сегмента PSP
- программный модуль

Приведем фрагмент дампа оперативной памяти, полученный при отладке программы MI.COM с помощью отладчика Microsoft CodeView. Дамп памяти начинается с адреса 6D47:0000.

Первая строка дампа (смещение 000-00F) - это МСВ блока памяти переменных среды. Это не последний блок памяти, поэтому МСВ имеет тип М. В поле владельца блока памяти находятся нули, следовательно, МСВ принадлежит сам себе. Поле длины содержит значение 0014 - это количество параграфов в блоке памяти переменных среды.

Со смещением 010 начинается блок переменных среды. Из дампа видно, что строки определения переменных среды закрыты двоичным нулем. Таблица переменных среды также закрыта нулем (смещение 13B). Со смещением 13C записано слово 0001 (количество слов в последующей строке), после которого расположен полный путь для файла запущенной программы.

```
000 4D00 0014 0000 0000 0000 0000 0000 0000 M.....
010 434F 4D53 5045 433D 433A 5C43 4F4D 4D41 COMSPEC=C:\COMMA
020 4E44 2E43 4F4D 0054 4D50 3D67 3A5C 7465 ND.COM.TMP=g:\te
030 6D70 0050 4154 483D 673A 5C3B 643A 5C71 mp.PATH=g:\;d:\q
040 6332 5C62 696E 3B65 3A5C 6336 3030 5C62 c2\bin;e:\c600\b
050 696E 623B 653A 5C63 3630 305C 6269 6E3B inb;e:\c600\bin;
060 633A 5C64 6F73 3B63 3A5C 6172 633B 663A c:\dos;c:\arc;f:
070 5C6E 6F72 746F 6E3B 653A 5C77 6F72 643B \norton;e:\word;
080 004C 4942 3D65 3A5C 7163 325C 4C49 4200 .LIB=e:\qc2\LIB.
090 494E 434C 5544 453D 673A 5C69 6E63 6C75 INCLUDE=g:\inclu
0A0 6465 3B65 3A5C 7163 325C 494E 434C 5544 de;e:\qc2\INCLUD
0B0 453B 653A 5C63 746F 6F6C 735C 696E 636C E;e:\ctools\incl
0C0 7564 653B 0048 454C 5046 494C 4553 3D65 ude;.HELPPFILES=e
0D0 3A5C 4336 3030 5C48 454C 505C 2A2E 484C :\C600\HELP\*.HL
0E0 503B 0049 4E49 543D 653A 5C43 3630 305C P;.INIT=e:\C600\
0F0 494E 4954 0044 4D41 4B45 3D67 3A5C 646D INIT.DMAKE=g:\dm
```



```

100 616B 653B 0056 4547 413D 653A 5C76 6567 ake;.VEGA=e:\veg
110 613B 004E 4152 4348 454C 503D 663A 5C61 a;.NARCHELP=f:\a
120 7263 5C6E 6172 633B 0048 454C 5050 4154 rc\narc;.HELPPAT
130 483D 643A 5C68 656C 703B 0000 0100 433A H=d:\help;....C:
140 5C44 4F53 5C6D 692E 636F 6D00 6D00 7016 \DOS\mi.com.m.p.

```

Далее начинается МСВ программного блока памяти (смещение 150). Это последний блок памяти, поэтому МСВ имеет тип Z. Непосредственно за МСВ располагается префикс программного сегмента PSP (смещение 160). Размер PSP - 256 байт, его формат будет описан ниже.

И, наконец, со смещением 260 расположена сама программа MI.COM.

```

150 5A00 00A3 3216 0000 6D69 0000 0000 0000 Z...2...mi.....
160 CD20 00A0 009A F0FE 1DF0 3D09 B22E 340A .....=...4.
170 B22E 850E B22E CD26 FFFF FFFF FFFF FFFF .....&.....
180 FFFF FFFF FFFF FFFF FFFF FFFF 486D 92B3 .....Hm..
190 FD4B 1400 1800 5D6D FFFF FFFF 0000 0000 .K....]m.....
1A0 0000 0000 0000 0000 0000 0000 0000 .....
1B0 CD21 CB00 0000 0000 0000 0000 0020 2020 .!.....
1C0 2020 2020 2020 2020 0000 0000 0020 2020 .....
1D0 2020 2020 2020 2020 0000 0000 0000 0000 .....
1E0 0320 2F61 0038 0F00 2C09 0001 0000 4005 . /a.8...@.
1F0 7808 0000 9000 0100 04B3 0D07 ED2F 14B3 x...../..
200 7501 4A36 0000 0000 0000 4003 A308 3F00 u.J6.....@...?.
210 FD4B FD4B 0100 2ABA 0000 0000 026F 5718 .K.K.*.....oW.
220 3800 0500 3800 CF08 DD26 2D09 FD4B FD4B 8...8....&-..K.K
230 E408 0000 4200 A308 3F00 FD4B FD4B E408 ....B...?..K.K..
240 1304 0000 5718 C000 0500 C000 CE02 CE03 ....W.....
250 5718 8400 0500 8400 CE02 CE03 973A 92B3 W.....
260 E9BB 000D 0A4D 656D 6F72 7920 496E 666F .....Memory Info
270 2076 352E 380D 0A43 6F70 7972 6967 6874 v5.8..Copyright
280 2031 3938 3920 4365 6E74 7261 6C20 506F 1989 Central Po
290 696E 7420 536F 6674 7761 7265 2C20 496E int Software, In
2A0 632E 2020 416C 6C20 7269 6768 7473 2072 c. All rights r
2B0 6573 6572 7665 642E 0D0A 0A00 4279 2047 eserved.....By G
2C0 5744 2030 312F 3036 2F38 391A 2D2D 2D2D WD 01/06/89.----
2D0 5041 5443 4820 4152 4541 2D2D 2D2D 2D2D PATCH AREA-----
2E0 2D2D FF90 5601 4102 0000 0290 5D6D 5C6D --..V.A.....]m\m

```

Префикс программного сегмента всегда создается при загрузке программы COM или EXE в память и имеет одинаковый формат для COM и EXE файлов:

- | | | | |
|-------|---|----------|---|
| (0) | 2 | int20h | - двоичный код команды int 20h (программы могут использовать эту команду для завершения своей работы) |
| (+2) | 2 | mem_top | - нижняя граница доступной памяти в системе в параграфах |
| (+4) | 1 | reserv1 | - зарезервировано |
| (+5) | 5 | call_dsp | - команда вызова FAR CALL диспетчера MS-DOS |
| (+10) | 4 | term_adr | - адрес завершения (Terminate Address) |
| (+14) | 4 | cbrk_adr | - адрес обработчика CTRL+BREAK |

(+18)	4	crit_err	- адрес обработчика критической ошибки
(+22)	2	parn_psp	- сегмент PSP программы, запустившей данную программу (программы-родителя)
(+24)	20	file_tab	- таблица открытых файлов, если здесь находятся байты 0FFH, то таблица не используется
(+44)	2	env_seg	- сегмент блока памяти, содержащего переменные среды
(+46)	4	ss_sp	- адрес стека SS:SP программы
(+50)	2	max_open	- максимальное число открытых файлов
(+52)	4	file_tba	- адрес таблицы открытых файлов
(+56)	24	reserv2	- зарезервировано
(+80)	3	disp	- диспетчер функций DOS
(+83)	9	reserv3	- зарезервировано
(+92)	16	fcbl	- форматируется как стандартный FCB, если первый аргумент командной строки содержит правильное имя файла
(+108)	20	fcbl2	- заполняется для второго аргумента командной строки аналогично fcbl
(+128)	1	p_size	- число значащих символов в неформатированной области параметров либо буфер обмена с диском DTA, назначенный по умолчанию
(+129)	127	parm	- неформатированная область параметров, заполняется при запуске программы из командной строки

Для обращения к полям PSP файл sysp.h содержит следующее определение:

```
#pragma pack(1)
```

```
typedef struct _PSP_ {
    unsigned char int20h[2];
    unsigned mem_top;
    unsigned char reserv1;
    unsigned char call_dsp[5];
    void far *term_adr;
    void far *cbrk_adr;
    void far *crit_err;
    unsigned parn_psp;
    unsigned char file_tab[20];
    unsigned env_seg;
    void far *ss_sp;
    unsigned max_open;
    void far *file_tba;
    unsigned char reserv2[24];
    unsigned char disp[3];
}
```



```

    unsigned char reserv3[9];
    unsigned char fcb1[16];
    unsigned char fcb2[20];
    unsigned char p_size;
    unsigned char parm[127];
} PSP;

```

```
#pragma pack()
```

Приведем программу, определяющую адрес первого блока MCB:

```

/**
*.Name      get_fmcb
*.
*.Title     Получить адрес первого MCB
*.
*.Descr     Функция возвращает адрес первого блока MCB
*.
*.Params    MCB far *get_fmcb(CVT far *cvt)
*.
*.           cvt - адрес векторной таблицы связи
*.
*.Return     Указатель на первый блок MCB
**/

```

```
#include "sysp.h"
```

```

MCB far *get_fmcb(CVT far *cvt)
{
    return((MCB far *)FP_MAKE(cvt->mcb_seg,0));
}

```

В качестве аргумента функции `get_fmcb` необходимо передать указатель на векторную таблицу связи, полученный с помощью описанной ранее функции `get_mcv`. Для получения адреса следующего MCB можно использовать функцию `get_nmcb`:

```

/**
*.Name      get_nmcb
*.
*.Title     Получить адрес следующего MCB
*.
*.Descr     Функция возвращает адрес следующего блока MCB или 0,
*.           если это последний блок. В качестве параметра
*.           используется указатель на предыдущий блок MCB
*.
*.Params    MCB far *get_fmcb(MCB far *mcb)
*.
*.           mcb - адрес предыдущего MCB
*.
*.Return     Указатель на следующий блок MCB или 0, если
*.           это последний MCB
**/

```

```

#include <dos.h>
#include "sysp.h"

```



```

MCB far *get_nmcbl(MCB far *mcb)
{
    unsigned seg, off;

    if(mcb->type == 'M') {
        seg = FP_SEG(mcb) + mcb->size + 1;
        off = FP_OFF(mcb);
        return((MCB far *) FP_MAKE(seg,off));
    }
    else return ((MCB far *)0);
}

```

В завершение описания блоков управления памятью приведем программу, выводящую некоторую информацию об активных МСВ. Эта программа определяет адрес векторной таблицы связи, получает из этой таблицы адрес первого МСВ и сканирует весь список МСВ.

```

#include <dos.h>
#include <stdio.h>
#include "sysp.h"

void main(void);

void main(void)
{
    CVT far *cvt;
    MCB far *mcb;
    printf("\nБлоки управления памятью (МСВ)"
           "\nCopyright (C)Frolov A., 1990\n"
           "\nАдрес МСВ Тип Владелец Размер"
           "\n-----"
           "\n");
    cvt=get_mcvl();
    mcb=get_fmcb(cvt);
    for(;;) {
        if(mcb == (MCB far *)0) break;
        printf("%Fp %c %04X %04X\n",
               mcb,
               mcb->type,
               mcb->owner,
               mcb->size);
        mcb=get_nmcbl(mcb);
    }
    exit(0);
}

```

2.3. Список управляющих блоков устройств

Поле dev_cb векторной таблицы связи содержит FAR-адрес цепочки блоков управления устройствами DOS (DOS Device Control Block) - DDCB. Блок DDCB строится операционной системой для каждого дискового устройства и содержит информацию о характеристиках этого устройства и указатель на заголовок драйвера, обслуживающего данное устройство.

Этот блок может быть использован программами, которые выполняют доступ к диску на уровне секторов. Подробнее назначение и использование полей блока DDCB будет описано в разделе, посвященном файловой системе, т. к. эта информация требуется в основном для организации работы с диском на низком уровне.

Приведем формат блока DDCB для MS-DOS версий 2.x и 3.x:

(0)	1	drv_num	- номер устройства (0 соответствует устройству A, 1 - B и т. д.)
(+1)	1	drv_numd	- дополнительный номер устройства внутри драйвера
(+2)	2	sec_size	- размер сектора в байтах
(+4)	1	clu_size	- число, на единицу меньшее количества секторов в кластере
(+5)	1	clu_base	- если содержимое этого поля не равно нулю, то для получения общего числа секторов в кластере надо возвести 2 в степень clu_base и получившееся число прибавить к clu_size
(+6)	2	boot_siz	- количество зарезервированных секторов (boot-секторы, начало корневого каталога)
(+8)	1	fat_num	- количество копий FAT
(+9)	2	max_dir	- максимальное число дескрипторов файлов в корневом каталоге (т. е. максимальное число файлов, которое может содержать корневой каталог на этом устройстве)
(+11)	2	data_sec	- номер первого сектора данных на диске (номер сектора, соответствующего кластеру номер 2)
(+13)	2	hi_clust	- максимальное количество кластеров (равно увеличенному на 1 количеству кластеров данных)
(+15)	1	fat_size	- количество секторов, занимаемых одной копией FAT
(+16)	2	root_sec	- номер первого сектора корневого каталога
(+18)	4	drv_addr	- FAR-адрес заголовка драйвера, обслуживающего данное устройство
(+22)	1	media	- байт описания среды носителя данных
(+23)	1	acc_flag	- флаг доступа, 0 - означает, что к устройству был доступ
(+24)	4	next	- адрес следующего блока DDCB, для последнего блока в поле смещения находится число FFFF

----- только для DOS 2.x -----

- | | | | |
|-------|----|----------|--|
| (+28) | 2 | dir_clu | - номер начального кластера текущего каталога (0 для корневого каталога) |
| (+30) | 64 | dir_path | - строка в формате ASCIIZ, содержащая путь к текущему каталогу |

----- DOS 3.x -----

- | | | | |
|-------|---|---------|---|
| (+28) | 2 | reserv1 | - зарезервировано, обычно равно 0 |
| (+30) | 2 | built | - число FFFF в этом поле означает, что блок DDCB был построен |

Для MS-DOS версии 4.x формат этого блока другой. Кроме того, изменилась его длина:

- | | | | |
|-------|---|----------|--|
| (0) | 1 | drv_num | - номер устройства (0 соответствует устройству A, 1 - B и т.д.) |
| (+1) | 1 | drv_numd | - дополнительный номер устройства внутри драйвера |
| (+2) | 2 | sec_size | - размер сектора в байтах |
| (+4) | 1 | clu_size | - число, на единицу меньшее количества секторов в кластере |
| (+5) | 1 | clu_base | - если содержимое этого поля не равно нулю, то для получения общего числа секторов в кластере надо возвести 2 в степень clu_base и получившееся число прибавить к clu_size |
| (+6) | 2 | boot_siz | - количество зарезервированных секторов (boot-секторы, начало корневого каталога) |
| (+8) | 1 | fat_num | - количество копий FAT |
| (+9) | 2 | max_dir | - максимальное число дескрипторов файлов в корневом каталоге (т. е. максимальное число файлов, которое может содержать корневой каталог на этом устройстве) |
| (+11) | 2 | data_sec | - номер первого сектора данных на диске (номер сектора, соответствующего кластеру номер 2) |
| (+13) | 2 | hi_clust | - максимальное количество кластеров (равно увеличенному на 1 количеству кластеров данных) |
| (+15) | 1 | fat_size | - количество секторов, занимаемых копией FAT |
| (+16) | 1 | reserv1 | - зарезервировано |
| (+17) | 2 | root_sec | - номер первого сектора корневого каталога |

- (+19) 4 drv_addr - FAR-адрес заголовка драйвера, обслуживающего данное устройство
- (+23) 1 media - байт описания среды носителя данных
- (+24) 1 acc_flag - флаг доступа: 0 - к устройству был доступ
- (+25) 4 next - адрес следующего блока DDCB, для последнего блока в поле смещения находится число FFFF
- (+29) 2 reserv2 - зарезервировано
- (+31) 2 built - число FFFF в этом поле означает, что блок DDCB был построен

Файл sysp.h содержит определение типа DDCB для MS-DOS версии 4.x:

/* Блок управления устройством DOS */

#pragma pack(1)

```
typedef struct _DDCB_ {
    unsigned char drv_num;
    unsigned char drv_numd;
    unsigned sec_size;
    unsigned char clu_size;
    unsigned char clu_base;
    unsigned boot_siz;
    unsigned char fat_num;
    unsigned max_dir;
    unsigned data_sec;
    unsigned hi_clust;
    unsigned char fat_size;
    char reserv1;
    unsigned root_sec;
    void far *drv_addr;
    unsigned char media;
    unsigned char acc_flag;
    struct _DDCB_ far *next;
    unsigned reserv2;
    unsigned built;
} DDCB;
```

#pragma pack()

Еще раз уместно заметить, что формат этого блока не описан в документации по MS-DOS и он может отличаться в различных версиях операционных систем.

Приведем тексты программ для получения адресов первого и последующих блоков DDCB:

```
/**
 * .Name            get_fddcb
 *
 * .Ti2tle          Получить адрес первого DDCB
 */
```



```

*.Descr      Функция возвращает адрес первого блока DDCB
*
*.Params      DDCB far *get_fddcb(CVT far *cvt)
*
*              cvt - адрес векторной таблицы связи
*
*.Return      Указатель на первый блок DDCB
**/

#include <stdlib.h>
#include <stdio.h>
#include "sysp.h"

DDCB far *get_fddcb(CVT far *cvt) {
    DDCB far * ddcb;
    ddcb = cvt->dev_cb;
    return(ddcb);
}

/**
*.Name        get_nddcb
*
*.Title       Получить адрес следующего DDCB
*
*.Descr       Функция возвращает адрес следующего блока DDCB
*              или 0, если это последний блок в цепочке
*
*.Params      DDCB far *get_nddcb(DDCB far *ddcb)
*
*              ddcb - адрес предыдущего DDCB
*
*.Return      Указатель на следующий блок DDCB
*              или 0, если это последний блок в цепочке
**/

#include <dos.h>
#include "sysp.h"

DDCB far *get_nddcb(DDCB far *ddcb) {
    DDCB far *ddcb_n;

    ddcb_n = ddcb->next;
    if(FP_OFF(ddcb_n) == 0xffff) return((DDCB far *)0);
    return(ddcb_n);
}

```

С помощью приведенной ниже программы можно просмотреть содержимое всех блоков DDCB. Т. к. при большом количестве дисков выводится очень много информации, следует использовать средство переназначения стандартного устройства вывода DOS:

```
show_ddc > drives.lst
```


Эта программа проверена для версии MS-DOS 4.01.

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "sysp.h"

void main(void);

void main(void)
{
    CVT far *cvt;
    DDCB far *ddcb;

    printf("\nБлоки управления дисковыми устройствами (DDCB)"
           "\nCopyright (C)Frolov A., 1990\n"
           "\n");

    cvt=get_mcv();
    ddcb=get_fddcb(cvt);

    for(;;) {
        if(ddcb == (DDCB far *)0) break;
        printf("Адрес DDCB: %Fp\n"
               "Номер устройства: %d\n"
               "Дополнительный номер: %d\n"
               "Размер сектора: %d\n"
               "Размер кластера в секторах: %d\n"
               "База размера кластера: %d\n"
               "Зарезервировано секторов: %d\n"
               "Число копий FAT: %d\n"
               "Макс. файлов в корневом каталоге : %d\n"
               "Первый кластер данных: %d\n"
               "Всего кластеров: %d\n"
               "Размер FAT в секторах: %d\n"
               "Первый сектор корневого каталога: %d\n"
               "Поле reserv1: %01X\n"
               "Адрес драйвера: %Fp\n"
               "Байт описателя среды носителя: %01X\n"
               "Флаг доступа: %01X\n"
               "Адрес следующего DDCB: %Fp\n"
               "Поле reserv2: %04X\n"
               "Блок построен: %04X\n"
               "-----\n\n",
               ddcb,
               ddcb->drv_num,
               ddcb->drv_numd,
               ddcb->sec_size,
               ddcb->clu_size,
               ddcb->clu_base,
               ddcb->boot_siz,
               ddcb->fat_num,
               ddcb->max_dir,
               ddcb->data_sec,
               ddcb->hi_clust,
```



```

        ddcb->fat_size,
        ddcb->root_sec,
        ddcb->reserv1,
        ddcb->drv_addr,
        ddcb->media,
        ddcb->acc_flag,
        ddcb->next,
        ddcb->reserv2,
        ddcb->built);
    ddcb=get_nddcb(ddcb);
}
    exit(0);
}

```

Приведенный выше способ получения доступа к блокам DDCB больше всего подходит для просмотра блоков управления всеми дисковыми устройствами. Если вам требуется получить DDCB для какого-нибудь конкретного устройства, можно воспользоваться недокументированной функцией 32H прерывания INT 21H (со всеми ограничениями, связанными с использованием недокументированных возможностей).

Функция 32H получает в регистре DL номер устройства (0 - текущий диск, 1 - A и т. д.) и возвращает в регистровой паре DS:BX адрес соответствующего DDCB. Если номер устройства был задан неправильно, регистр AL после выполнения функции будет содержать значение FF.

Если требуется получить адрес DDCB флоппи-диска, необходимо установить диск в приемный карман дисковода. Приведем текст программы, возвращающей указатель на DDCB диска с заданным номером:

```

/**
*.Name      get_ddcb
*.
*.Title     Получить адрес DDCB заданного диска
*.
*.Descr     Функция возвращает адрес блока управления
*.           устройством DOS DDCB
*.
*.Params    DDCB far *get_ddcb(int device_number)
*.
*.           device_number - номер диска, для которого требуется
*.                           получить DDCB. Номер задается так:
*.                           Ø - текущий диск, 1 - B и т.д.
*.
*.Return     Указатель на DDCB заданного диска
**/

```

```

#include <dos.h>
#include <stdio.h>
#include "sysp.h"

```

```

DDCB far *get_ddcb(unsigned char device_number) {
    union REGS inregs, outregs;
    struct SREGS segregs;

```



```

inregs.h.ah = 0x32;
inregs.h.al = 0;
inregs.h.dl = device_number;
intdosx( &inregs, &outregs, &segregs );
if(outregs.h.al == 0xff) return(DDCB far *)0;

return((DDCB far*)FP_MAKE(segregs.ds,outregs.x.bx));
}

```

Программа, приведенная ниже, выводит адреса всех DDCB. Можете запустить ее (она есть на дискете, прилагающейся к книге) и посмотреть, что получится. Не забудьте вставить флоппи-диски во все дисководы.

```

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "sysp.h"

void main(void);

void main(void)
{
    DDCB far *ddcb;
    unsigned char dr;

    for(dr=1;;dr++) {
        ddcb=get_ddcb(dr);
        if(ddcb == (DDCB far *)0) break;
        printf("%Fp\n",ddcb);
    }
    exit(0);
}

```

2.4. Таблица файлов MS-DOS

DOS создает таблицу открытых файлов и помещает ее адрес в поле `file_tab` векторной таблицы связи. В этой таблице для каждого открытого файла хранится такая информация, как количество файловых чисел (file handle), связанных с ним, режим открытия файла (чтение, запись и т. д.), слово информации об устройстве, указатель на заголовок драйвера, обслуживающего данное устройство, элемент дескриптора файла (дата, время, имя файла, номер начального кластера, распределенного файлу), номер последнего прочитанного кластера и т. д.

Эта информация может пригодиться при организации защиты программы от копирования путем ее привязки к номерам занимаемых программой кластеров. Заметьте, что получить информацию о начальном кластере файла довольно трудно - стандартные средства DOS не предоставляют такой возможности. Приходится работать с диском на уровне секторов, отслеживать FAT, читать напрямую каталог и т. д. Таблица файлов содержит этот номер в явном виде.

Строка файла CONFIG.SYS может содержать оператор FILES=xx. Этот оператор в конечном счете определяет размер таблицы файлов DOS (DFT). Каждая

таблица DFT содержит указатель на следующую таблицу и количество управляющих блоков файлов DOS (DFCB). Сами блоки DFCB (по одному для каждого файла) расположены в конце таблицы DFT.

Формат этого блока различается для DOS 3.x и 4.x.

Приведем сначала формат таблицы файлов для DOS 3.x:

(0)	4	next	- указатель на следующую таблицу файлов
(+4)	2	file_count	- количество файлов в этой таблице
--- Дальше идут блоки DFCB в количестве file_count штук ----			
(0)	2	handl_num	- количество файловых чисел, связанных с данным файлом (file handle)
(+2)	1	access_mode	- режим доступа к файлу, заданный при открытии файла
(+3)	2	reserv1	- зарезервировано
(+5)	2	dev_info	- информация IOCTL, полученная для устройства, на котором расположен этот файл (подробно формат и назначение этого поля будут рассмотрены в главе, посвященной драйверам)
(+7)	4	driver	- указатель на драйвер, обслуживающий устройство, содержащее файл
(+11)	2	first_clu	- номер первого кластера, распределенного файлу
(+13)	2	time	- время последнего изменения файла в упакованном формате
(+15)	2	date	- дата последнего изменения файла в упакованном формате
(+17)	4	fl_size	- размер файла в байтах
(+21)	4	offset	- текущее смещение внутри файла в байтах
(+25)	2	reserv2	- зарезервировано
(+27)	2	last_clu	- номер только что прочитанного кластера
(+29)	3	reserv3	- зарезервировано
(+32)	11	filename	- имя файла в формате FCB: выровнено на левую границу поля, дополнено пробелами до 8 символов, справа от него - 3 символа расширения без точки
(+43)	2	reserv4	- зарезервировано
(+45)	2	ownr_psp	- PSP программы, открывшей файл
(+47)	2	reserv5	- зарезервировано

Операционная система MS-DOS версии 4.x отличается расположением поля last_clu, кроме того, изменилась длина DFCB:

- | | | | |
|------|---|------------|---|
| (0) | 4 | next | - указатель на следующую таблицу файлов |
| (+4) | 2 | file_count | - количество файлов в этой таблице |

--- Дальше идут блоки DFCB в количестве file_count штук ----

- | | | | |
|-------|----|-------------|---|
| (0) | 2 | handl_num | - количество файловых чисел, связанных с данным файлом (file handle) |
| (+2) | 1 | access_mode | - режим доступа к файлу, заданный при открытии файла |
| (+3) | 2 | reserv1 | - зарезервировано |
| (+5) | 2 | dev_info | - информация IOCTL, полученная для устройства, на котором расположен этот файл (подробно формат и назначение этого поля будут рассмотрены в главе, посвященной драйверам) |
| (+7) | 4 | driver | - указатель на драйвер, обслуживающий устройство, содержащее файл |
| (+11) | 2 | first_clu | - номер первого кластера, распределенного файлу |
| (+13) | 2 | time | - время последнего изменения файла в упакованном формате |
| (+15) | 2 | date | - дата последнего изменения файла в упакованном формате |
| (+17) | 4 | fl_size | - размер файла в байтах |
| (+21) | 4 | offset | - текущее смещение внутри файла в байтах |
| (+25) | 2 | reserv2 | - зарезервировано |
| (+27) | 2 | reserv7 | - зарезервировано |
| (+29) | 3 | reserv3 | - зарезервировано |
| (+32) | 1 | reserv4 | - зарезервировано |
| (+33) | 11 | filename | - имя файла в формате FCB: выравнено на левую границу поля, дополнено пробелами до 8 символов, справа от него - 3 символа расширения без точки |
| (+44) | 2 | reserv5 | - зарезервировано |
| (+46) | 2 | ownr_psp | - PSP программы, открывшей файл |
| (+48) | 2 | reserv6 | - зарезервировано |
| (+50) | 2 | last_clu | - номер только что прочитанного кластера |
| (+52) | 4 | reserv8 | - зарезервировано |

Для версии MS-DOS 4.01 файл sysp.h содержит определение структур для работы с таблицами файлов:

```
typedef struct _DFCB_ {
    unsigned handl_num;
    unsigned char access_mode;
    unsigned reserv1;
    unsigned dev_info;
    void far *driver;
    unsigned first_clu;
    unsigned time;
    unsigned date;
    unsigned long fl_size;
    unsigned long offset;
    unsigned reserv2;
    unsigned reserv7;
    unsigned reserv3;
    char reserv4;
    char filename[11];
    char reserv5[6];
    unsigned ownr_psp;
    unsigned reserv6;
    unsigned last_clu;
    char reserv8[4];
} DFCB;

typedef struct _DFT_ {
    struct _DFT_ far *next;
    unsigned file_count;
    DFCB dfcb;
} DFT;
```

Приведем текст программ, возвращающих указатели на первый и последующий элементы списка таблиц файлов DOS:

```
/**
 * .Name      get_fdft
 *
 * .Title     Получить адрес первой DTF
 *
 * .Descr     Функция возвращает адрес первой таблицы файлов DOS
 *
 * .Params    DTF far *get_fdtf(CVT far *cvt)
 *
 *            cvt - адрес векторной таблицы связи
 *
 * .Return    Указатель на первый блок DDCB
 */

#include <stdlib.h>
#include <stdio.h>
#include "sysp.h"

DFT far *get_fdft(CVT far *cvt) {
    DFT far * dft;
```



```

    dft = cvt->file_tab;
    return(dft);
}

/**
*.Name      get_ndft
*.Title     Получить адрес следующей DTF
*.Descr     Функция возвращает адрес следующей таблицы
*.          файлов DOS или 0, если это последняя таблица
*.Params    DFT far *get_ndft(DFT far *dft)
*.          dft - адрес предыдущей таблицы DFT
*.Return    Указатель на следующую DFT или 0, если последняя
**/
#include <dos.h>
#include <stdlib.h>
#include <stdio.h>
#include "sysp.h"

DFT far *get_ndft(DFT far *dft) {
    DFT far * dft_next;

    dft_next = dft->next;
    if(FP_OFF(dft_next) == 0xffff) return((DFT far *)0);

    return(dft_next);
}

```

Для подробной распечатки содержимого таблицы файлов можно использовать следующую программу, которая была проверена в MS-DOS версии 4.01:

```

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "sysp.h"

void main(void);

void main(void)
{
    CVT far *cvt;
    DFT far *dft;
    unsigned i,j,k;
    DFCB far *dfcb;
    FILE *list;

    printf("Информация об открытых файлах DOS\n"
           "Copyright Frolov A. (C),1990\n");
}

```



```

// Открываем файл для вывода информации о файлах
list=fopen("!dfcb.lst","w+");

fprintf(list,"Информация об открытых файлах DOS\n"
          "Copyright Frolov A. (C),1990\n\n");

cvt=get_mcvf(); // Адрес векторной таблицы связи
dft=get_fdft(cvt); // Адрес начала таблицы файлов

for(;;) {
    if(dft == (DDCB far *)0) break; // Конец таблицы

    i=dft->file_count;
    fprintf(list,"Таблица файлов DFT: %Fp, в ней %d"
              "файлов\n"
              "=====\n",
            dft,i);

    for(j=0;j<i;j++) { // Цикл по файловым
                        // управляющим блокам

        dfcb=(&(dft->dfcb))+j; // Адрес DFcb файла

        fprintf(list,"\nDFcb файла: %Fp\n\n",dfcb);

        fprintf(list,"Имя файла: ");
        for(k=0;k<11;k++) {
            fputc(dfcb->filename[k],list);
        }

        fprintf(list,"\nКоличество file handles: %d\n"
                  "Режим доступа: %d\n"
                  "Поле reserv1: %04X\n"
                  "Информация об устройстве: %04X\n"
                  "Адрес драйвера: %Fp\n"
                  "Начальный кластер: %d\n"
                  "Время: %04X\n"
                  "Дата: %04X\n"
                  "Размер файла в байтах: %ld\n"
                  "Текущее смещение в файле: %ld\n"
                  "Поле reserv2: %04X\n"
                  "Последний прочитанный кластер: %d\n"
                  "Сегмент PSP владельца файла: %04X\n"
                  "Поле reserv7: %d\n"
                  "-----\n\n",
                  dfcb->handl_num,
                  dfcb->access_mode,
                  dfcb->reserv1,
                  dfcb->dev_info,
                  dfcb->driver,
                  dfcb->first_clu,
                  dfcb->time,
                  dfcb->date,
                  dfcb->fl_size,
                  dfcb->offset,

```



```

                                dfcb->reserv2,
                                dfcb->last_clu,
                                dfcb->ownr_psp,
                                dfcb->reserv7);
        }
        dft=get_ndft(dft);
    }
    fclose(list);
    exit(0);
}

```

Описание содержимого таблицы файлов будет записано в файл с именем "dfcb.lst".

2.5. Список загружаемых драйверов устройств

Все драйверы, резидентные или подключенные к операционной системе во время обработки файла CONFIG.SYS связаны в список. Первый драйвер - это так называемый NUL-драйвер - располагается всегда непосредственно после векторной таблицы связи. Подробно о драйверах будет рассказано во второй книге, а сейчас мы кратко опишем, как проследить цепочку загруженных драйверов и получить некоторые сведения об этих драйверах.

Драйвер - это программа, которая занимает не более одного сегмента (64 Кбайта) и имеет в самом начале специальный заголовок - заголовок драйвера следующего формата:

(0)	4	next	- указатель на заголовок следующего драйвера. Если смещение адреса следующего драйвера равно FFFF, это последний драйвер в цепочке
(+4)	2	attrib	- атрибуты драйвера
(+6)	2	strateg	- смещение программы стратегии драйвера
(+8)	2	interrupt	- смещение программы обработки прерывания для драйвера
(+10)	8	dev_name	- для символьных устройств - имя устройства, для блочных - количество обслуживаемых устройств

Мы приведем программу, которая сканирует список драйверов и выводит на стандартное устройство вывода адрес драйвера, его атрибуты, имя устройства для символьных устройств и количество обслуживаемых устройств для блочных драйверов. Эта программа написана на языке Ассемблера.

; Программа выводит информацию о загруженных драйверах

; Макроопределение печатает символы на экране

```

@@out_ch MACRO c1,c2,c3,c4,c5
    mov     ah,02h
    IRP     chr,<c1,c2,c3,c4,c5>

```



```

        IFB    <chr>
        EXITM
        ENDIF
        mov    dl,chr
        int    21h
        ENDM
        ENDM

        .MODEL tiny
        DCSSEG

        .STACK 100h

        .DATA

msg      DB 13,10,"Device Drivers Information V1.00", 13, 10
        DB      "Copyright (C)Frolov A .1990",13,10,13,10
        DB      "Address      Attr  Device Name",13,10
        DB      "-----      ----  -----",13,10
        DB "$"

bl_msg   DB "-----> Block Device, Number of Units: ", "$"

        .CODE
        .STARTUP

        mov    ah, 9h                ; Выводим заголовок
        mov    dx, OFFSET msg
        int    21h

        mov    ah,52h                ; Получаем адрес первого
        int    21h                    ; драйвера в цепочке
        add    bx,22h                 ; es:bx - адрес первого драйвера

dr_loop:
        cmp    WORD PTR es:[bx+2], 0ffffh ; последний ?
        jz     end_of_driver_list
        cmp    WORD PTR es:[bx], 0ffffh
        jz     end_of_driver_list

        call   show_driver_info        ; выводим параметры драйвера

        mov    ax,es:[bx]              ; получаем адрес следующего
        mov    cx,es:[bx+2]            ; драйвера
        mov    bx,ax
        mov    es,cx

        jmp    dr_loop

end_of_driver_list:

        EXIT    0

show_driver_info proc near                ; es:bx - адрес драйвера
        push    es
        push    bx

```



```

        mov     ax,es                      ; выводим адрес драйвера
        call    Print_word
        @@out_ch ':'
        mov     ax,bx
        call    Print_word
        @@out_ch ','

        mov     ax,es:[bx+4]              ; выводим атрибут драйвера
        call    Print_word
        @@out_ch ','

        test    WORD PTR es:[bx+4],8000h  ; проверяем: это
                                           ; символьный
        jz      is_block                  ; драйвер или блочный

        mov     cx,8                      ; для символьного выводим
        mov     si,bx                    ; имя драйвера
pr_name:
        mov     al,BYTE PTR es:[si+10]
        @@out_ch al
        inc     si
        loop    pr_name
        jmp     nxt

is_block:

        mov     ah, 9h                   ; для блочного драйвера
        mov     dx, OFFSET bl_msg        ; выводим количество
        int     21h                      ; логических устройств,
        mov     al,BYTE PTR es:[bx+10]   ; которые обслуживает
        mov     ah,0                     ; этот драйвер
        call    Print_word

nxt:

        @@out_ch 13,10

        pop     bx
        pop     es

        ret
show_driver_info endp

; Вывод на экран содержимого регистра AX
Print_word proc near
;-----
        push    ax
        push    bx
        push    dx
;
        push    ax
        mov     cl,8
        rol     ax,cl
        call    Byte_to_hex
        mov     bx,dx

```



```
    @@out_ch bh
    @@out_ch bl
;
    pop ax
    call Byte_to_hex
    mov bx,dx
    @@out_ch bh
    @@out_ch bl
;
    pop dx
    pop bx
    pop ax
    ret
Print_word endp
;
Byte_to_hex proc near
;-----
; al - input byte
; dx - output hex
;-----
    push ds
    push cx
    push bx
;
    lea bx,tabl
    mov dx,cs
    mov ds,dx
;
    push ax
    and al,0fh
    xlat
    mov dl,al
;
    pop ax
    mov cl,4
    shr al,cl
    xlat
    mov dh,al
;
    pop bx
    pop cx
    pop ds
    ret
;
tabl db '0123456789ABCDEF'
Byte_to_hex endp
;
END
```


Если запустить эту программу, она выведет на экран сведения о всех загруженных драйверах:

```
Device Drivers Information V1.00
Copyright (C)Frolov A.,1990
```

Address	Attr	Device Name
02C1:0048	8004	NUL
112F:0000	8800	RBUSDRIV
10E4:0000	0800	-----> Block Device, Number of Units: 0001
0D86:0000	C800	SMARTAAR
0CC7:0000	A000	XMSXXXX0
0BA5:0000	6842	-----> Block Device, Number of Units: 0003
0070:016E	8013	CON
0070:0180	8000	AUX
0070:0192	A040	PRN
0070:01A4	8008	CLOCK\$
0070:01B6	0842	-----> Block Device, Number of Units: 0003
0070:01CA	8000	COM1
0070:01DC	A040	LPT1
0070:01EE	A040	LPT2
0070:0200	A040	LPT3
0070:0212	8000	COM2
0070:0224	8000	COM3

К этой картинке мы еще вернемся при обсуждении процесса загрузки драйверов.

2.6. Другие поля векторной таблицы связи

Как мы только что увидели, все драйверы связаны в цепочку, которую нетрудно проследить. Для драйверов часов CLOCK\$ и консоли CON векторная таблица связи содержит их актуальные адреса, соответственно в полях clock_dr и con_dr. Если вам надо получить адреса актуальных драйверов, то самый быстрый способ - взять эти адреса из векторной таблицы связи.

Поле lastdriv содержит значение команды LASTDRIVE в файле CONFIG.SYS. Его можно использовать для определения максимального количества дисковых устройств в системе. Количество действительно используемых блочных устройств находится в поле num_bdev.

Если CONFIG.SYS содержит команду FCBS=xx, то в поле fcb_tabl находится адрес таблицы FCB, а в поле fcb_size - размер этой таблицы.

В поле max_btbl находится максимальное число байтов, содержащихся в блоке блочного устройства.

Очень интересные сведения находятся в массиве дисковой информации, на который указывает поле drv_info векторной таблицы связи. Количество элементов этого массива равно количеству дисковых устройств в системе.

Для каждого диска в этом массиве можно найти текущий путь доступа в виде строки ASCIIZ, указатель на блок управления устройствами DOS DDCB, номер начального кластера текущего каталога (кстати, если к какому-либо каталогу ни разу не обращались, в этом поле находится -1).

Приведем формат элемента массива и соответствующую структуру в файле sysp.h:

(0)	64	path	- текущий путь доступа для диска
(+64)	2	reserv1	- зарезервировано
(+66)	2	reserv2	- зарезервировано
(+68)	1	reserv3	- зарезервировано
(+69)	4	ddcb	- адрес соответствующего DDCB
(+73)	2	cdir_clu	- первый кластер текущего каталога на диске. 0 соответствует корневому каталогу, -1 - если к диску еще не обращались
(+75)	2	reserv4	- зарезервировано
(+77)	2	reserv5	- зарезервировано
(+79)	2	reserv6	- зарезервировано

----- DOS 4.x -----

(+81)	7	reserv7	- зарезервировано
-------	---	---------	-------------------

Приводимое ниже определение типа DINFO соответствует формату MS-DOS 4.x.

```
#pragma pack(1)
```

```
typedef struct _DINFO_ {
    char path[64];
    unsigned reserv1;
    unsigned reserv2;
    unsigned char reserv3;
    DDCB far *ddcb;
    unsigned cdir_clu;
    unsigned reserv4;
    unsigned reserv5;
    unsigned reserv6;
    unsigned char reserv7[7];
} DINFO;
```

```
#pragma pack()
```

Как пример использования этой информации приведем программу, которая выводит содержимое массива на экран:

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "sysp.h"

void main(void);

void main(void)
{
    CVT far *cvt;
```



```

unsigned i,j,k;

printf("Информация о дисковых устройствах\n"
      "Copyright Frolov A. (C),1990\n");

cvt=get_mcv();      // Адрес векторной таблицы связи
dinfo=cvt->drv_info; // Адрес таблицы дисковых устройств
i=cvt->num_bdev;     // Количество дисковых устройств

for(j=0;j<i;j++) {
    printf("Адрес: %Fp, путь: %Fs\n"
          "Первый кластер каталога: %d\n\n",
          dinfo,
          dinfo->path,
          dinfo->cdir_clu);
    dinfo = dinfo+1;
}
exit(0);
}

```

На этом мы завершаем изучение векторной таблицы связи MS-DOS. Еще раз уместно напомнить, что вся приведенная выше информация отсутствует в документации и может меняться (и действительно меняется!) от версии к версии. Если вы хотите использовать описанные выше средства, вам необходимо динамически определять версию DOS и использовать соответствующие форматы управляющих блоков.

Продолжение, начало см. стр. 35

ПРАКТИЧЕСКАЯ РАБОТА НА КОМПЬЮТЕРАХ СЕМЕЙСТВА IBM PC В ОПЕРАЦИОННОЙ СРЕДЕ MS-DOS 4.01

Ю.Я. Максимов, С.В. Осипов, О.С. Симоненков

Приводятся общие сведения о персональных компьютерах (ПК) и операционных системах. Рассматриваются файловая система, основные команды MS-DOS и приемы работы с ней, а также операционная оболочка MS-DOS Shell.

Объем 8 л. Цена 13 руб.

ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ Microsoft BASIC

Ю.Я. Максимов и др.

Пособие содержит введение в язык Microsoft BASIC. Даны сравнительный анализ реализаций языка BASIC, описание языка Microsoft BASIC версий 4.0 и 4.5. Пособие предназначено для широкого круга пользователей ПЭВМ семейства IBM.

Объем 10 л. Цена 24 руб.

Продолжение на стр. 98, 220, 236

УПРАВЛЕНИЕ ПРОГРАММАМИ

3.1. Форматы программных файлов

Теперь, когда мы знаем структуру памяти на момент завершения загрузки операционной системы, можно посмотреть, а что же происходит дальше, когда оператор запускает какую-нибудь программу.

Оператор может запустить два типа программ (если не считать командных файлов, которые, вообще говоря, не являются программами, состоящими из машинных кодов) - программы, имеющие расширение имени .COM и .EXE. Эти файлы имеют различный формат и загружаются по-разному, однако когда загрузка завершена, в памяти компьютера эти два типа программ выглядят совершенно одинаково.

COM-файл - это двоичный образ вашей программы, состоящий из кода и данных. То есть это файл, содержащий программу в "чистом" виде. Такая программа (как и EXE-программа) может загружаться в любое место памяти. DOS выполняет ее привязку к физическим адресам при загрузке с помощью установки сегментных регистров. Существенным ограничением COM-программы является то, что она не может занимать больше одного сегмента (соответственно файл .COM не может быть по длине больше 64 Кбайт).

Программа в формате EXE может иметь любой размер. В самом начале файла программы содержится заголовок (у COM-файла заголовка нет). Этот заголовок используется операционной системой в процессе загрузки программы в память для правильной установки сегментных регистров. Заголовок EXE-файла нужен только при загрузке; когда программа загружена и готова к работе, самого заголовка уже нет в памяти.

Заголовок EXE-файла состоит из форматированной зоны и таблицы расположения сегментов (Relocation Table). Форматированная зона выглядит следующим образом:

(0)	2	signature	- два байта 'MZ' (4Dh, 5Ah), идентифицирующие файл в формате EXE
(+2)	2	part_pag	- длина последней страницы программы в байтах (страница содержит 512 байт)
(+4)	2	file_size	- размер программы в страницах по 512 байт
(+6)	2	rel_item	- число элементов в таблице расположения сегментов
(+8)	2	hdr_size	- размер заголовка файла в параграфах (длина параграфа - 16 байт)

- | | | | |
|-------|---|----------|---|
| (+10) | 2 | min_mem | - минимальное количество памяти в параграфах, которое нужно зарезервировать в памяти за концом загруженной программы |
| (+12) | 2 | max_mem | - максимальное количество памяти в параграфах, которое нужно зарезервировать в памяти за концом загруженной программы |
| (+14) | 2 | ss_reg | - величина смещения от начала программы, используемая для загрузки сегментного регистра стека SS |
| (+16) | 2 | sp_reg | - величина смещения от начала программы, которая используется для загрузки регистра SP |
| (+18) | 2 | chk_summ | - контрольная сумма всех слов в файле |
| (+20) | 2 | ip_reg | - значение для регистра IP, которое будет использовано при начальном запуске программы |
| (+22) | 2 | cs_reg | - смещение от начала программы для установки сегментного регистра кода CS |
| (+24) | 2 | reft_off | - смещение от начала файла таблицы расположения сегментов программы |
| (+26) | 2 | overlay | - номер оверлея, равен 0 для основного модуля |

Таблица расположения сегментов программы начинается сразу после форматированной области и состоит из четырехбайтовых значений в формате "смещение:сегмент".

Область файла после таблицы расположения сегментов выравнивается на границу параграфа с помощью байта-заполнителя и дальше начинается сама программа.

В файле sysp.h есть описание заголовка файла и таблицы расположения сегментов, которые вы можете использовать при обработке заголовка EXE-файла:

```
typedef struct _EXE_HDR_ {
    unsigned signature;
    unsigned part_pag;
    unsigned file_size;
    unsigned rel_item;
    unsigned hdr_size;
    unsigned min_mem;
    unsigned max_mem;
    unsigned ss_reg;
    unsigned sp_reg;
    unsigned chk_summ;
    unsigned ip_reg;
    unsigned cs_reg;
    unsigned reft_off;
    unsigned overlay;
} EXE_HDR;
```



```
typedef struct _RELOC_TAB_ {
    unsigned offset;
    unsigned segment;
} RELOC_TAB;
```

В качестве примера приведем программу, которая считывает форматированную часть заголовка EXE-файла, проверяет наличие в его первых двух байтах признака EXE-формата ('MZ'). Если признак имеется, программа выводит на экран расшифрованное содержимое заголовка и таблицу перемещений, если такая таблица присутствует. В качестве параметра программе надо при запуске передать имя исследуемого EXE-файла.

```
#include <stdio.h>
#include <stdlib.h>
#include "sysp.h"

void main(int, char *[]);

void main(int argc, char *argv[]) {
    printf("Распечатка заголовка EXE-файла\n"
           "Copyright (C)Frolov A., 1990\n\n");

    if( argc != 2 ) {
        printf( "  Задайте путь EXE-файла в качестве"
                "параметра\n" );
        exit(0);
    }

    if( gethdr( argv[1] ) != 0 ) {
        printf( "Ошибка в формате файла или нет такого"
                "файла\n" );
        exit(0);
    }
    exit(0);
}

int gethdr( char *path) {
    EXE_HDR    header;
    RELOC_TAB  *reloc;
    FILE       *inpf;
    int        i;

    if((inpf = fopen(path,"rb")) == 0) return(-1);

    if(get_exeh(&header,&reloc,inpf) != 0) {
        fclose(inpf);
        return(-1);
    }
    printf("Магическое число:           %04X\n"
           "Длина последней страницы файла:      %d\n"
           "Количество страниц в файле:             %d\n"
           "Кол. элементов табл. перемещений:         %d\n",
           header.magic, header.last_page, header.page_count, header.reloc_count);
}
```



```

"Размер заголовка в параграфах:           %d\n"
"Минимальная память для программы:        %04X\n"
"Максимальная память для программы:        %04X\n"
"Значение адреса стека SS:SP:               04X:%04X\n"
"Контрольная сумма:                        %04X\n"
"Значения для регистров CS:IP:               %04X:%04X\n"
"Смещение табл. перемещений:                %02X\n"
"Номер оверлея:                             %d\n",
    header.signature,
    header.part_pag,
    header.file_size,
    header.rel_item,
    header.hdr_size,
    header.min_mem,
    header.max_mem,
    header.ss_reg,
    header.sp_reg,
    header.chk_summ,
    header.cs_reg,
    header.ip_reg,
    header.relt_off,
    header.overlay);

```

```

if(reloc != 0) {
    printf("\nСодержимое таблицы перемещений:\n\n");

    for(i=0; i < header.rel_item; i++) {
        printf("%04X:%04X\n",
            (reloc+i)->segment,
            (reloc+i)->offset);
    }
    free(reloc);
}
fclose(inpfile);
return(0);
}

```

Приведенная выше программа для чтения заголовка EXE-файла пользуется функцией get-exeh:

```

/*
*.Name      get_exeh
*.Title     Прочитать заголовок EXE-файла
*.Descr     Функция читает заголовок EXE-файла в структуру типа
             EXE_HDR, заказывает память для таблицы размещений
             сегментов и считывает таблицу в эту область.
             Адрес заказанной области помещается по адресу,
             передаваемому в rtb. Если таблица размещений
             отсутствует, память для нее не заказывается.
*.Params    int get_exeh(EXE_HDR *exeh, RELOC_TAB **rtb,
             FILE *exe_file)
             exeh - указатель на структуру, которая

```



```

*           должна быть заполнена информацией
*           из заголовка EXE-файла
*
*           rtb - указатель на указатель на таблицу
*                размещений сегментов программы
*
*           exe_file - указатель на открытый EXE-файл
*                      (до вызова функции нельзя обращаться
*                      к этому файлу, т. к. считается, что
*                      указатель текущего смещения
*                      установлен на начало файла)
*
*..Return   0 при успешном считывании заголовка;
*          -1 в случае неправильного формата заголовка
**/

#include <stdlib.h>
#include <stdio.h>
#include "sysp.h"

int get_exeh(EXE_HDR *exeh, RELOC_TAB **rtb, FILE *exe_file) {
    int i, j, k;
    // считываем форматированную часть заголовка
    for(i=0; i < sizeof(EXE_HDR); i++) {
        *(((char*)exeh) + i) = fgetc(exe_file);
        if(feof(exe_file)) break;
    }
    // это EXE-файл?
    if(exeh->signature != 0x5a4d) return(-1);
    if((i=exeh->rel_item) != 0) {
        // если есть таблица перемещений, заказываем для нее память
        *rtb = (RELOC_TAB*)malloc(i*sizeof(RELOC_TAB)+16);
        // считываем таблицу перемещений
        for(k=0; k<i; k++) {
            for(j=0; j < sizeof(RELOC_TAB); j++) {
                *(((char*)(*rtb)+j+k*sizeof(RELOC_TAB)))=
                    fgetc(exe_file);
                if(feof(exe_file)) break;
            }
        }
    }
    else *rtb = (RELOC_TAB *)0;
    return(0);
}

```


3.2. Процесс загрузки программ в память

Загрузка COM- и EXE-программ происходит по-разному, однако есть некоторые действия, которые операционная система выполняет в обоих случаях одинаково.

- Определяется наименьший сегментный адрес свободного участка памяти для загрузки программы (обычно DOS загружает программу в младшие адреса памяти, если при редактировании не указана загрузка в старшие адреса).
- Создаются два блока памяти (и следовательно, два блока MCB, описанные ранее) - блок памяти для переменных среды и блок памяти для PSP и программы.
- Для MS-DOS версии 3.x и старше в блок памяти переменных среды помещается путь файла программы.
- Заполняются поля префикса сегмента программы PSP в соответствии с характеристиками программы (количество памяти, доступное программе, адрес сегмента блока памяти, содержащего переменные среды и т. д.)
- Устанавливается адрес области Disk Transfer Area (DTA) на вторую половину PSP (PSP:0080).
- Анализируются параметры запуска программы на предмет наличия в первых двух параметрах идентификаторов дисковых устройств. По результатам анализа устанавливается содержимое регистра AX при входе в программу. Если первый или второй параметры не содержат правильного идентификатора дискового устройства, то соответственно в регистры AL и AH записывается значение FF.

А дальше действия системы по загрузке программ форматов COM и EXE будут различаться.

Для COM-программ, которые представляют собой двоичный образ односегментной программы, выполняется чтение файла программы с диска и запись его в память по адресу PSP:0100. Вообще говоря, программы типа COM могут состоять из нескольких сегментов, но в этом случае они должны сами управлять содержимым сегментных регистров, используя в качестве базового адреса адрес PSP.

После загрузки файла операционная система для COM-программ выполняет следующие действия:

- сегментные регистры CS, DS, ES, SS устанавливаются на начало PSP
- регистр SP устанавливается на конец сегмента PSP
- вся область памяти после PSP распределяется программе
- в стек записывается слово 0000
- указатель команд IP устанавливается на 100h (начало программы) с помощью команды JMP по адресу PSP:100

Загрузка EXE-программ происходит значительно сложнее, т. к. связана с настройкой сегментных адресов:

- Считывается во внутренний буфер DOS форматированная часть заголовка файла.
- Определяется размер загрузочного модуля по формуле:
$$\text{size} = (\text{file_size} * 512) - (\text{hdr_size} * 16) - \text{part_pag}$$
- Определяется смещение начала загрузочного модуля в EXE-файле как $\text{hdr_size} * 16$.
- Вычисляется сегментный адрес для загрузки START_SEG, обычно используется значение $\text{PSP} + 10\text{h}$.
- Загрузочный модуль считывается в память по адресу START_SEG:0000.
- Сканируются элементы таблицы перемещений, располагающейся в EXE-файле со смещением reft_off .
- Для каждого элемента таблицы:
 1. Считывается содержимое элемента таблицы как два двухбайтных слова (OFF, SEG).
 2. Вычисляется сегментный адрес ссылки перемещения
$$\text{REL_SEG} = (\text{START_SEG} + \text{SEG})$$
 3. Выбирается слово по адресу REL_SEG:OFF, к нему прибавляется значение START_SEG, затем сумма записывается обратно по тому же адресу.
- Заказывается память для программы, исходя из значений min_mem и max_mem .
- Инициализируются регистры и программа запускается на выполнение. При инициализации регистры ES и DS устанавливаются на PSP, регистр AX устанавливается так же, как и для COM-программ, в сегментный регистр стека SS записывается значение $\text{START_SEG} + \text{ss_reg}$, а в SP записывается sp_reg .

Для запуска программы в CS записывается $\text{START_SEG} + \text{cs_reg}$, а в IP - ip_reg . Такая запись невозможна напрямую, поэтому операционная система сначала записывает в свой стек значение для CS, затем значение для IP и после этого выполняет команду дальнего возврата RETF (команда возврата из дальней процедуры).

3.3. Префикс программного сегмента

Теперь займемся плотную префиксом программного сегмента PSP. Формат PSP уже был описан ранее, для удобства приведем его еще раз вместе со структурой из файла `sysp.h`:

- (0) 2 `int20h` - двоичный код команды `int 20h` (программы могут использовать эту команду для завершения своей работы)

- | | | | |
|--------|-----|----------|---|
| (+2) | 2 | mem_top | - нижняя граница доступной памяти в системе в пара-
графах |
| (+4) | 1 | reserv1 | - зарезервировано |
| (+5) | 5 | call_dsp | - команда вызова FAR CALL диспетчера MS-DOS |
| (+10) | 4 | term_adr | - адрес завершения (Terminate Address) |
| (+14) | 4 | cbrk_adr | - адрес обработчика CTRL+BREAK |
| (+18) | 4 | crit_err | - адрес обработчика критической ошибки |
| (+22) | 2 | parn_psp | - сегмент PSP программы, запустившей данную прог-
рамму (программы-родителя) |
| (+24) | 20 | file_tab | - таблица открытых файлов, если здесь находятся бай-
ты 0FFH, то таблица не используется |
| (+44) | 2 | env_seg | - сегмент блока памяти с переменными среды |
| (+46) | 4 | ss_sp | - адрес стека SS:SP программы |
| (+50) | 2 | max_open | - максимальное число открытых файлов |
| (+52) | 4 | file_tba | - адрес таблицы открытых файлов |
| (+56) | 24 | reserv2 | - зарезервировано |
| (+80) | 3 | disp | - диспетчер функций DOS |
| (+83) | 9 | reserv3 | - зарезервировано |
| (+92) | 16 | fcbl | - форматируется как стандартный FCB, если первый
аргумент командной строки содержит правильное
имя файла |
| (+108) | 20 | fcbl | - заполняется для второго аргумента командной стро-
ки аналогично fcbl |
| (+128) | 1 | p_size | - число значащих символов в неформатированной об-
ласти параметров либо буфер обмена с диском DTA,
назначенный по умолчанию |
| (+129) | 127 | parm | - неформатированная область параметров, заполняет-
ся при запуске программы из командной строки |

```
#pragma pack(1)
```

```
typedef struct _PSP_ {  
    unsigned char int20h[2];  
    unsigned mem_top;  
    unsigned char reserv1;  
    unsigned char call_dsp[5];  
    void far *term_adr;  
    void far *cbrk_adr;  
    void far *crit_err;
```



```

unsigned parn_psp;
unsigned char file_tab[20];
unsigned env_seg;
void far *ss_sp;
unsigned max_open;
void far *file_tba;
unsigned char reserv2[24];
unsigned char disp[3];
unsigned char reserv3[9];
unsigned char fcb1[16];
unsigned char fcb2[20];
unsigned char p_size;
unsigned char parm[127];
} PSP;

```

```
#pragma pack()
```

Программы могут получить из PSP такую информацию, как параметры командной строки при запуске, размер доступной памяти, найти сегмент области переменных среды и т. д.

Как программе узнать адрес своего PSP? Очень просто сделать это для программ, написанных на языке Ассемблера: при запуске программы этот адрес передается ей через регистры DS и ES. Т. е. этот адрес равен DS:0000 или ES:0000 (для COM-программ на PSP указывают также регистры CS и SS).

Для программ, составленных на языке Си, доступна глобальная переменная `_psp` типа `unsigned`. Эта переменная содержит сегментный адрес PSP.

В качестве примера приведем текст программы на языке Ассемблера, которая выводит на экран передаваемые ей через PSP параметры запуска:

```

.MODEL tiny
DOSSEG

.STACK 100h

.DATA

parm_msg DB "Укажите параметры", 13, 10, "$"

.CODE
.STARTUP

mov     cl,ds:80h      ; количество символов
                        ; в командной строке
cmp     cl,0
je      ask_parm      ; нет параметров - просим
                        ; задать параметры

mov     si,81h         ; со смещением 81h начинается
                        ; область параметров
cld

get_parm:

lods    BYTE PTR es:[si] ; загружаем в al очередной
                        ; символ строки параметров

```



```

mov     ah,2                ; выводим его на экран
mov     dl,al
int     21h

loop    get_parm
jmp     end_progr

```

```
ask_parm:
```

```

mov     ah, 9h
mov     dx, OFFSET parm_msg
int     21h

```

```
end_progr:
.EXIT   0
```

```
END
```

Приведенная ниже программа, составленная на языке Си, определяет адрес своего PSP, затем показывает содержимое некоторых полей из PSP:

```

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "sysp.h"

void main(void);

void main(void) {
    PSP far *psp_ptr;

    psp_ptr = FP_MAKE(_psp,0); // Конструируем указатель на PSP
    printf("PSP расположено по адресу:      %Fp\n"
           "Доступно памяти, байт:          %ld\n"
           "PSP родительской программы:      %Fp\n"
           "\n",
           psp_ptr,
           (long)(psp_ptr->mem_top)*16L,
           FP_MAKE(psp_ptr->parn_psp,0));
    exit(0);
}

```

Используя поле `parn_psp`, можно определить адрес PSP родительской программы, т. е. программы, запустившей вашу программу.

Немного о назначении полей `term_adr`, `cbrk_adr`, `crit_err`.

Поле `term_adr` содержит значение, полученное из таблицы векторов прерываний для вектора 22h. Это адрес программы, которая получает управление, когда текущая программа завершает свою работу. Это может быть, например, `COMMAND.COM`. Программа может создать свою собственную подпрограмму, которая будет получать управление при завершении работы основной программы. Она может записать свой собственный адрес в вектор 22h, затем запустить другую программу. В таком случае в запущенной программе это поле в ее PSP будет содержать адрес родительской программы. Когда основная программа завершает свою работу, DOS восстанавливает адрес программы завершения в векторе 22h из поля `term_adr` PSP.

Поле `cbrk_adr` содержит адрес программы обработки прерывания по нажатию `CTRL+BREAK` из вектора `23h` таблицы векторов прерываний. Т. к. программа может устанавливать свою собственную программу обработки прерывания по `CTRL+BREAK`, DOS при завершении работы программы восстанавливает оригинальное значение из поля `cbrk_adr`.

Аналогично поле `crit_err` предназначено для восстановления содержимого вектора `24h` - адреса обработчика критических ошибок.

Способы переназначения векторов будут приведены в разделе, посвященном прерываниям.

Конечно, программы, составленные на языке Си, не обязательно должны использовать PSP для доступа к параметрам командной строки и переменным среды. Для этого есть параметры функции `main` и набор функций типа `getenv`, `putenv` и т. п., предназначенных для работы со средой. Но ведь PSP содержит и другую информацию!

3.4. Запуск программ из программ

Ваша программа может при необходимости запустить другую программу формата EXE или COM. Для ассемблерных программ существует функция `4Bh` прерывания `INT 21h`, для программ, составленных на языке Си, - разнообразные функции, входящие в состав стандартной библиотеки. Сначала рассмотрим запуск программ при помощи функции `4Bh` прерывания `INT 21h`.

Содержимое регистров перед вызовом прерывания:

AX = 4BH

AL - код подфункции (0, 1, 2, 3)

DS:DX - указатель на путь к запускаемой программе

ES:BX - указатель на блок параметров EPB

После возврата из прерывания флаг `CF` устанавливается в 0, если ошибок не было, и в 1 при обнаружении ошибок. Регистр `AX` в случае наличия ошибок содержит код ошибки:

- 1 - неверный код подфункции
- 2 - файл запускаемой программы не найден
- 3 - путь не найден
- 4 - слишком много открытых файлов
- 5 - нет доступа
- 8 - нет памяти для загрузки программы
- 10 - длина блока среды больше 32 килобайт
- 11 - плохой формат запускаемого EXE-файла

Функция `4Bh` прерывания `21h` имеет четыре подфункции с номерами от 0 до 3:

- 0 - загрузить и выполнить программу
- 1 - загрузить, но не выполнять программу (внутренняя подфункция для MS-DOS 3.x)

- 2 - загрузить, но не выполнять программу (внутренняя подфункция для DOS 2.x);
- 3 - загрузить программу как оверлей (не создавать PSP).

Для функции 0 регистры DS:DX должны указывать на полный путь запускаемой программы в формате ASCIIZ (т. е. текстовая строка, закрытая двоичным нулем). Блок параметров EPB (Exec Parameter Block) в этом случае имеет следующий формат:

- | | | | |
|-------|---|---------|---|
| (0) | 2 | seg_env | - сегментный адрес среды, создаваемой родительской программой для запускаемой программы. Если в этом поле находится 0, то для запускаемой программы копируется среда родительской программы |
| (+2) | 4 | cmd | - FAR-адрес строки параметров для запускаемой программы. Эта строка должна иметь такой же формат, как и в PSP, т. е. вначале идет байт со значением, равным количеству символов в строке параметров, а затем - сама строка параметров |
| (+6) | 4 | fcb1 | - адрес блока FCB, помещаемого в PSP со смещением 5Ch (в PSP помещается блок, а не адрес!) |
| (+10) | 4 | fcb2 | - адрес блока FCB, который будет помещен в PSP со смещением 6Ch. |

Запущенной программе доступны все файлы, открытые родительской программой.

Если родительская программа сама формирует среду для дочерней программы, она должна подготовить новую среду на границе параграфа и поместить значение сегментного адреса в поле seg_env блока EPB.

Приведем простую программу, которая запускает программу с именем PARM.COM из текущего каталога. Программу PARM.COM мы только что рассматривали, эта программа выводит на экран полученные ею в командной строке параметры.

```

        .MODEL    small
        DOSSEG

        .STACK    1000h

        .DATA

path          db "PARM.COM",0
command_line  db 8,"Parm Str"

epb           dw 0
cmd_off       dw ?
cmd_seg       dw ?
fcb1          dd ?
fcb2          dd ?

```



```

.CODE
.STARTUP

mov     bx,OFFSET command_line ; адрес командной
mov     cmd_off,bx             ; строки для блока EPB
mov     cmd_seg,ds

mov     ax,ds
mov     es,ax

mov     bx,OFFSET epb ; ES:BX указывают на EPB
mov     dx,OFFSET path ; DS:DX указывают на путь
                        ; запускаемой программы

mov     ax, 4B00h ; AH = 4Bh
                        ; AL = 0 загрузить и выполнить
int     21h

.EXIT   0

END

```

Эта программа использует модель памяти Small, и ее загрузочный модуль имеет формат EXE. При редактировании был указан стандартный для QuickC 2.01 размер памяти, требуемый для программы. Если попытаться использовать формат COM в модели Tiny, то окажется, что вся память распределена COM-программе и для дочерней программы не осталось места.

Следующая программа освобождает всю неиспользуемую ей память, после чего на освободившееся место загружает программу PARM.COM:

```

.MODEL tiny
DOSSEG

.STACK 100h

.DATA

path      db "PARM.COM",0
command_line db 8,"Parm Str"

epb       dw 0
cmd_off   dw ?
cmd_seg   dw ?
fcb1      dd ?
fcb2      dd ?

.CODE
.STARTUP

;
; Освобождаем лишнюю память за концом программы
;

mov     bx,OFFSET last ; смещение конца программы

mov     cl,4
shr     bx,cl           ; вычисляем длину в параграфах

```



```

        add     bx,17             ; добавляем 1 параграф для
                                ; выравнивания и 256 байт
                                ; для стека

        mov     ah, 4Ah           ; изменяем размер выделенного
int     21h                     ; блока памяти

        mov     ax,bx             ; установка нового значения
        shl     ax,cl             ; указателя стека
        dec     ax
        mov     sp,ax

        mov     bx,OFFSET command_line ; адрес командной
        mov     cmd_off,bx        ; строки для блока EPB
        mov     cmd_seg,ds

        mov     ax,ds
        mov     es,ax

        mov     bx,OFFSET epb    ; ES:BX указывают на EPB
        mov     dx,OFFSET path   ; DS:DX указывают на путь
                                ; запускаемой программы

        mov     ax, 4B00h        ; AH = 4Bh
                                ; AL = 0 загрузить и выполнить
int     21h

        .EXIT     0
last:    db ?
        END

```

Для изменения размера выделенного программе блока памяти мы использовали функцию 4Ah прерывания 21h.

Подфункции 1 и 2 прерывания 4Bh используются DOS (это внутренние подфункции DOS). Для них мы приведем недокументированный формат блока EBP.

Для подфункции 1:

- | | | | |
|-------|---|---------|---|
| (0) | 2 | seg_env | - сегментный адрес среды, создаваемой родительской программой для запускаемой программы. Если в этом поле находится 0, то для запускаемой программы копируется среда родительской программы |
| (+2) | 4 | cmd | - FAR-адрес строки параметров для запускаемой программы |
| (+6) | 4 | fcbl | - адрес блока FCB, который будет помещен в PSP со смещением 5Ch |
| (+10) | 4 | fcbl | - адрес блока FCB, который будет помещен в PSP со смещением 6Ch. |
| (+14) | 4 | ss_sp | - это поле будет содержать значение SS:SP после возврата |
| (+18) | 4 | entry_p | - адрес точки входа в загруженную программу (CS:IP) |


```
int      21h
.EXIT    Ø
buff:    dd 100 dup(?)
END
```

Программа загружается в буфер buff.

Пользователи языка Си имеют в своем распоряжении три возможности запустить программу.

Самый простой способ - использовать функцию `system()`, которая может выполнить любую команду DOS или любую программу, пакетный файл. Например:

```
system("FORMAT A:");
```

При использовании этой функции должен быть доступен `COMMAND.COM`. К сожалению, хотя `system` и возвращает код завершения, по нему нельзя сделать вывод о том, как выполнялась запускаемая программа. Если в качестве аргумента функции будет передано неправильное имя, на экране появится сообщение:

```
Bad command or file name
```

Код возврата в этом случае будет 0 - как будто все нормально!

Другие две возможности запустить программу - использовать функции `spawn` и `exec`. Функция `spawn` и ее разновидности запускают программу как дочерний процесс. Функция `exec` загружает новую программу как оверлей на место старой и передает ей управление без возврата. После завершения дочерней программы управление будет передано `COMMAND.COM` или программе, которая запустила родительскую программу.

Семейство функций `spawn` обеспечивает запуск дочерней программы с родительской или со сформированной специально средой. Кроме того, в файле `process.h` описаны параметры, которые можно передать функции `spawn`:

`P_WAIT` - выполнение родительской программы задерживается до завершения дочерней программы.

`P_NOWAIT` - родительская программа продолжает выполнение сразу после запуска дочерней. Этот параметр имеет смысл только для операционных систем `OS/2`, `UNIX`, в которых поддерживается мультизадачность.

`P_OVERLAY` - загружает программу как оверлей и передает ей управление. Этот режим соответствует функции `exec` в том смысле, что родительская программа не получит управление после завершения дочерней.

В качестве примера использования функций запуска программы рассмотрим возможное решение задачи создания `HELP`-системы для прикладной программы.

С помощью текстового редактора можно создать справочную базу данных в формате утилиты `Microsoft HELPMAKE`, затем, запуская в нужный момент диалоговую утилиту работы с базой данных `Microsoft Quick Help QH.EXE`, можно получить нужную справку.

Утилита `QH` использует базы данных, описанные в переменной среды `HELPPFILES`. Мы будем использовать либо родительскую среду, где находится

значение переменной HELPFILES по умолчанию, либо указывать новое значение для этой переменной.

Приведенная ниже программа используется для получения справки о функции стандартной библиотеки printf, поиск производится в HELP-базе QuickC:

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
```

```
main()
{
    int r;

    // Получаем справку о функции printf, справочная база
    // данных расположена в каталоге d:\qc2\bin
    r = help("HELPFILES=d:\\qc2\\bin;", "printf");
    if( r == -1 )
        printf( "Невозможно запустить процесс" );
    else
        printf( "\nПроцесс завершен" );
    exit(r);
}
```

```
/**
*.Name      help
*.Title     Получить справку по заданному контексту
*.Descr     Функция получает в качестве параметров переменную
*.           среды, указывающую на путь к справочной базе
*.           данных и указатель на строку контекста для поиска
*.           в базе. Затем запускается как дочерний процесс
*.           утилиты Microsoft Quick Help QH.EXE, для
*.           которой формируются среда и параметры.
*.
*.Params    int help(char *help_file, char *help_topic);
*.           help_file - переменная среды, указывающая
*.                   на путь к справочной базе
*.           help_topic - контекст для поиска в базе
*.
*.Return     0 при успешном запуске процесса
*.           -1 не удалось запустить процесс
**/
```

```
int help(char *help_file, char *help_topic) {
    char *env[] = { "", NULL }; // Среда, которую
                                // получит QH при запуске

    if(*help_file != 0) {
        env[0] = help_file; // Формируем среду для QH
    }
}
```



```
        // Запускаем утилиту
        return(spawnlpe(P_WAIT, "QH", "QH",
            "-u", help_topic, NULL, env));
    }
    else {
        // Если переменная среды не задана,
        // используем родительскую среду
        return(spawnlp(P_WAIT, "QH", "QH",
            "-u", help_topic, NULL));
    }
}
```

Подробная информация об использовании утилит HELPMAKE и QH приводится в документации на Microsoft C 6.0.

3.5. Завершение работы программы

Старые версии MS-DOS (до 2.0) требовали выполнения достаточно сложной процедуры для завершения программы. В начале работы программы необходимо было сохранить адрес PSP. Затем перед завершением работы поместить этот адрес в стек, поместить в стек слово 0000 и выполнить команду дальнего возврата. Управление при этом передается в начало PSP, где находится команда INT 20h.

Для версий MS-DOS начиная с 2.0 существуют более удобные способы - использование напрямую команды INT 20h или функции 0 прерывания 21h (CS при этом должен указывать на PSP, поэтому этот способ хорош для COM-программ) или функции 4Ch прерывания 21h в любое время и с любым содержимым регистров.

Последний способ рекомендуется для использования и имеет еще то преимущество, что позволяет передать родительской программе (например, COMMAND.COM) код завершения. Этот код доступен для анализа в пакетных файлах командой IF ERRORLEVEL.

Приведенные в книге примеры программ на языке Ассемблера содержат директиву .EXIT. Эта директива завершает выполнение программы с помощью функции 4Ch и позволяет передать код завершения.

Если ваша программа запустила дочернюю программу и та завершилась с передачей кода возврата, то родительская программа может определить этот код с помощью функции 4Dh прерывания 21h. Эта функция возвращает код в регистре AX.

Программа, написанная на языке Си, может завершаться с помощью return в функции main или с помощью exit в любом месте программы. При этом также возможна передача кода возврата.

Существуют еще способы завершения работы программы, при которых программа (или ее часть) остается резидентной в памяти. Это вызов прерывания INT 27h или функции 31h прерывания INT 21h. Об этом будет подробно рассказано в разделе, посвященном резидентным программам.

ПРЕРЫВАНИЯ

4.1. Механизм прерываний

Для обработки событий, происходящих асинхронно по отношению к выполнению программы, лучше всего подходит механизм прерываний. Прерывание можно рассматривать как некоторое особое событие в системе, требующее моментальной реакции. Например, хорошо спроектированные системы повышенной надежности используют прерывание по аварии в питающей сети для выполнения процедур записи содержимого регистров и оперативной памяти на магнитный носитель, с тем чтобы после восстановления питания можно было продолжить работу с того же места.

Кажется очевидным, что возможны самые разнообразные прерывания по самым различным причинам. Поэтому прерывание рассматривается не просто как таковое, с ним связывают число, называемое номером типа прерывания или просто номером прерывания. С каждым номером прерывания связывается то или иное событие. Система умеет распознавать, какое прерывание, с каким номером произошло и запускает соответствующую этому номеру процедуру.

Программы могут сами вызывать прерывания с заданным номером. Для этого они используют команду INT. Это так называемые программные прерывания. Программные прерывания не являются асинхронными, т. к. вызываются из программы (а она-то знает, когда она вызывает прерывание!).

Программные прерывания удобно использовать для организации доступа к отдельным, общим для всех программ модулям. Например, программные модули операционной системы доступны прикладным программам именно через прерывания, и нет необходимости при вызове этих модулей знать их текущий адрес в памяти. Прикладные программы могут сами устанавливать свои обработчики прерываний для их последующего использования другими программами. Для этого встраиваемые обработчики прерываний должны быть резидентными в памяти. Мы научимся создавать свои программы обработки прерываний и будем говорить об этом при обсуждении резидентных программ.

Аппаратные прерывания вызываются физическими устройствами и приходят асинхронно. Эти прерывания информируют систему о событиях, связанных с работой устройств, например о том, что наконец-то завершилась печать символа на принтере и неплохо было бы выдать следующий символ, или о том, что требуемый сектор диска уже прочитан, его содержимое доступно программе.

Использование прерываний при работе с медленными внешними устройствами позволяют совместить ввод/вывод с обработкой данных в центральном процессоре и в результате повышает общую производительность системы.

Некоторые прерывания (первые пять в порядке номеров) зарезервированы для использования самим центральным процессором на случай каких-либо особых событий вроде попытки деления на ноль, переполнения и т. п.

Иногда желательно сделать систему нечувствительной ко всем или отдельным прерываниям. Для этого используют так называемое маскирование прерываний, о котором мы еще будем подробно говорить. Но некоторые прерывания замаскировать нельзя, это немаскируемые прерывания.

Заметим еще, что обработчики прерываний могут сами вызывать программные прерывания, например для получения доступа к сервису BIOS или DOS (сервис BIOS также доступен через механизм программных прерываний).

Составление собственных программ обработки прерываний и замена стандартных обработчиков DOS и BIOS является ответственной и сложной работой. Необходимо учитывать все тонкости работы аппаратуры и взаимодействия программного и аппаратного обеспечения. При отладке возможно разрушение операционной системы с непредсказуемыми последствиями, поэтому надо очень внимательно следить за тем, что делает ваша программа.

4.2. Таблица векторов прерываний

Для того чтобы связать адрес обработчика прерывания с номером прерывания, используется таблица векторов прерываний, занимающая первый килобайт оперативной памяти - адреса от 0000:0000 до 0000:03FF. Таблица состоит из 256 элементов - FAR-адресов обработчиков прерываний. Эти элементы называются векторами прерываний. В первом слове элемента таблицы записано смещение, а во втором - адрес сегмента обработчика прерывания.

Прерыванию с номером 0 соответствует адрес 0000:0000, прерыванию с номером 1 - 0000:0004 и т. д. Для программиста, использующего язык Си, таблицу можно описать следующим образом:

```
void (* interrupt_table[256])();
```

Инициализация таблицы происходит частично BIOS после тестирования аппаратуры и перед началом загрузки операционной системой, частично при загрузке DOS. DOS может переключить на себя некоторые прерывания BIOS.

Займемся теперь содержанием таблицы векторов прерываний. Приведем на-значение некоторых наиболее важных векторов:

Номер	Описание
0	Ошибка деления. Вызывается автоматически после выполнения команд DIV или IDIV, если в результате деления происходит переполнение (например, при делении на 0). DOS обычно при обработке этого прерывания выводит сообщение об ошибке и останавливает выполнение программы. Для процессора 8086 при этом адрес возврата указывает на следующую после команды деления команду, а в процессоре 80286 - на первый байт команды, вызвавшей прерывание

- 1 Прерывание пошагового режима. Вырабатывается после выполнения каждой машинной команды, если в слове флагов установлен бит пошаговой трассировки TF. Используется для отладки программ. Это прерывание не вырабатывается после выполнения команды MOV в сегментные регистры или после загрузки сегментных регистров командой POP
- 2 Аппаратное немаскируемое прерывание. Это прерывание может использоваться по-разному в разных машинах. Обычно вырабатывается при ошибке четности в оперативной памяти и при запросе прерывания от сопроцессора
- 3 Прерывание для трассировки. Это прерывание генерируется при выполнении однобайтовой машинной команды с кодом CSh и обычно используется отладчиками для установки точки прерывания
- 4 Переполнение. Генерируется машинной командой INTO, если установлен флаг OF. Если флаг не установлен, то команда INTO выполняется как NOP. Это прерывание используется для обработки ошибок при выполнении арифметических операций
- 5 Печать копии экрана. Генерируется при нажатии на клавиатуре клавиши PrtScr. Обычно используется для печати образа экрана. Для процессора 80286 генерируется при выполнении машинной команды BOUND, если проверяемое значение вышло за пределы заданного диапазона
- 6 Неопределенный код операции или длина команды больше 10 байт (для процессора 80286)
- 7 Особый случай отсутствия математического сопроцессора (процессор 80286)
- 8 IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду
- 9 IRQ1 - прерывание от клавиатуры. Генерируется при нажатии и при отжатии клавиши. Используется для чтения данных от клавиатуры
- A IRQ2 - используется для каскадирования аппаратных прерываний в машинах класса AT
- B IRQ3 - прерывание асинхронного порта COM2
- C IRQ4 - прерывание асинхронного порта COM1
- D IRQ5 - прерывание от контроллера жесткого диска для XT
- E IRQ6 - прерывание генерируется контроллером флоппи-диска после завершения операции
- F IRQ7 - прерывание принтера. Генерируется принтером, когда он готов к выполнению очередной операции. Многие адаптеры принтера не используют это прерывание
- 10 Обслуживание видеоадаптера

11	Определение конфигурации устройств в системе
12	Определение размера оперативной памяти в системе
13	Обслуживание дисковой системы
14	Последовательный ввод/вывод
15	Расширенный сервис для АТ-компьютеров
16	Обслуживание клавиатуры
17	Обслуживание принтера
18	Запуск BASIC в ПЗУ, если он есть
19	Загрузка операционной системы
1A	Обслуживание часов
1B	Обработчик прерывания CTRL+BREAK
1C	Прерывание возникает 18,2 раза в секунду, вызывается программно обработчиком прерывания таймера
1D	Адрес видеотаблицы для контроллера видеоадаптера 6845
1E	Указатель на таблицу параметров дискеты
1F	Указатель на графическую таблицу для символов с кодами ASCII 128-255
20-5F	Используется DOS или зарезервировано для DOS
60-67	Прерывания, зарезервированные для пользователя
68-6F	Не используются
70	IRQ8 - прерывание от часов реального времени
71	IRQ9 - прерывание от контроллера EGA
72	IRQ10 - зарезервировано
73	IRQ11 - зарезервировано
74	IRQ12 - зарезервировано
75	IRQ13 - прерывание от математического сопроцессора
76	IRQ14 - прерывание от контроллера жесткого диска
77	IRQ15 - зарезервировано
78 - 7F	Не используются
80-85	Зарезервированы для BASIC
86-F0	Используются интерпретатором BASIC
F1-FF	Не используются

IRQ0-IRQ15 - это аппаратные прерывания, о них будет рассказано позже.

4.3. Маскирование прерываний

Часто при выполнении критических участков программ для того, чтобы гарантировать выполнение определенной последовательности команд целиком приходится запрещать прерывания. Это можно сделать командой CLI. Ее нужно поместить в начало критической последовательности команд, а в конце расположить команду STI, разрешающую процессору воспринимать прерывания. Команда CLI запрещает только маскируемые прерывания, немаскируемые всегда обрабатываются процессором. Если вы используете запрет прерываний с помощью команды CLI, следите за тем, чтобы прерывания не отключались на длительный период времени, т. к. это может привести к нежелательным последствиям, например будут отставать часы.

Если вам надо запретить не все прерывания, а только некоторые, например от клавиатуры, то для этого надо воспользоваться услугами контроллера прерываний. Подробно об этом немного ниже, сейчас отметим только, что выдачей в этот контроллер определенной управляющей информации можно замаскировать прерывания от отдельных устройств.

4.4. Изменение таблицы векторов прерываний

Вашей программе может потребоваться организовать обработку некоторых прерываний. Для этого программа должна переназначить вектор на свой обработчик. Это можно сделать, изменив содержимое соответствующего элемента таблицы векторов прерываний.

Очень важно не забыть перед завершением работы восстановить содержимое измененных векторов в таблице прерываний, т. к. после завершения работы программы память, которая была ей распределена, считается свободной и может быть использована для загрузки другой программы. Если вы забыли восстановить вектор и пришло прерывание, то система может разрушиться - вектор теперь указывает на область, которая может содержать что угодно. Поэтому последовательность действий для нерезидентных программ, желающих обрабатывать прерывания, должна быть такой:

- прочитать содержимое элемента таблицы векторов прерываний для вектора с нужным вам номером
- запомнить это содержимое (адрес старого обработчика прерывания) в области данных программы
- установить новый адрес в таблице векторов прерываний так, чтобы он соответствовал началу вашей программы обработки прерывания
- перед завершением работы программы прочитать из области данных адрес старого обработчика прерывания и записать его в таблицу векторов прерываний

Кроме того, операция изменения вектора прерывания должна быть непрерывной в том смысле, что во время изменения не должно произойти прерывание с номером, для которого производится замена программы обработки. Если вы,

например, запишете новое значение смещения, а сегментный адрес обновить не успеете, то по какому адресу будет передано управление в случае прерывания и что при этом произойдет? Об этом можно только догадываться.

Для облегчения работы по замене прерывания DOS предоставляет в ваше распоряжение специальные функции для чтения элемента таблицы векторов прерывания и для записи в нее нового адреса. Если вы будете использовать эти функции, DOS гарантирует, что операция по замене вектора будет выполнена правильно; вам не надо заботиться о непрерывности процесса замены вектора прерывания.

Для чтения вектора используйте функцию 35h прерывания 21h. Перед ее вызовом регистр AL должен содержать номер вектора в таблице. После выполнения функции в регистрах ES:BX будет искомый адрес обработчика прерывания.

Функция 25h прерывания 21h устанавливает для вектора с номером, находящимся в AL, обработчик прерывания DS:DX.

Разумеется, вы можете непосредственно обращаться к таблице векторов прерываний, но тогда при записи необходимо замаскировать прерывания командой CLI, не забыв разрешить их после записи командой STI.

Для пользователей языка Си библиотека QuickC содержит функции `_dos_getvec()`, `_dos_setvect()`. Первая функция получает адрес из таблицы векторов прерываний, вторая устанавливает новый адрес.

Если вам надо добавить какие-либо собственные действия к тем, что выполняет стандартный обработчик прерывания, то можно организовать цепочку прерываний.

Для организации цепочки прерываний вы записываете в векторную таблицу адрес своего обработчика, не забыв сохранить прежнее содержимое таблицы. Ваш обработчик получает управление по прерыванию, выполняет какие-либо действия, затем передает управление стандартному обработчику. Можно сделать и по-другому: ваш обработчик вызывает стандартный как подпрограмму, после возврата из стандартного обработчика выполняет дополнительные действия. То есть вы можете вставить дополнительную обработку как до вызова стандартного обработчика, так и после его вызова.

Для организации цепочки прерываний в библиотеке QuickC имеется функция `_chain_intr()`.

Рассмотрим более подробно возможности библиотеки интегрированной среды QuickC, предназначенные для работы с прерываниями.

Модификатор `interrupt` (`_interrupt` для QuickC 2.5 и C 6.0) описывает функцию, которая является обработчиком прерывания. Такая функция завершается командой возврата из обработки прерывания `IRET`, и для нее автоматически генерируются команды сохранения регистров на входе и их восстановления при выходе из обработчика прерывания. Пример использования модификатора для описания функции обработки прерывания:

```
void interrupt far int_func(void) {  
    // Тело обработчика прерывания  
}
```


Функция обработки прерывания должна быть FAR-функцией, т. к. таблица векторов прерываний содержит полные адреса в виде сегмент:смещение.

Ключевое слово `interrupt` используется также для описания переменных, предназначенных для хранения векторов прерываний:

```
void (_interrupt _far *oldvect)(void);
```

Модификаторы `_interrupt` и `_far` для QuickC 2.5 и C 6.0 являются синонимами соответственно `interrupt` и `far`.

Какие требования предъявляются к программе обработки прерывания?

Если прерывания происходят часто, то их обработка может сильно замедлить работу прикладной программы. Поэтому обработчик прерывания должен быть короткой, быстро работающей программой, которая выполняет только самые необходимые действия. Например, считать очередной символ из порта принтера и поместить его в буфер, увеличить значение какого-либо глобального счетчика прерываний и т. п.

Для установки своего обработчика прерываний используйте функцию `_dos_setvect`. Эта функция имеет два параметра - номер прерывания и указатель на новую функцию обработки прерывания. Например:

```
_dos_setvect(0x16, my_key_intr);
```

В этом примере для клавиатурного прерывания с номером `16h` устанавливается новый обработчик прерывания `my_key_intr`.

Если вам надо узнать адрес старого обработчика прерывания по его номеру, лучше всего воспользоваться функцией `_dos_getvect`, которая принимает в качестве параметра номер прерывания и возвращает указатель на соответствующий этому номеру в таблице векторов прерываний обработчик. Например:

```
old_vector = _dos_getvect(0x16);
```

Для организации цепочки прерываний используйте функцию `_chain_intr`. В качестве параметра эта функция принимает адрес старого обработчика прерываний.

Следующий простой пример иллюстрирует применение всех трех функций, предназначенных для работы с прерываниями. Эта программа встраивает собственный обработчик прерывания таймера, который будет вызываться примерно 18,2 раза в секунду. Встраиваемый обработчик прерывания считает тики таймера, и, если значение счетчика кратно 20, на динамик компьютера выдается звуковой сигнал. В конце работы новая программа обработки прерывания таймера вызывает старый обработчик с помощью функции `_chain_intr`.

После установки нового обработчика прерывания таймера основная программа ждет нажатия на клавиатуре любой клавиши, затем она восстанавливает старое содержимое вектора прерывания.

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
```



```
// Выключаем проверку стека и указателей

#pragma check_stack( off )
#pragma check_pointer( off )

// Это макро используется для выдачи сигнала на внутренний
// динамик компьютера. Используется вывод в формате TTY
// символа BELL (7) через прерывание BIOS 10h

#define BEEP() _asm { \
                    _asm mov bx,0           \
                    _asm mov ax, 0E07h      \
                    _asm int 10h            \
                }

void main(void);

// Объявление программы обработки прерывания

void _interrupt _far timer(void);

// Эта переменная предназначена для хранения старого значения
// вектора прерывания таймера. Она должна быть глобальной.

void (_interrupt _far *oldvect)(void);

// Переменная для подсчета тиков таймера
volatile long ticks;

void main(void) {
    ticks=0L; // Сбрасываем счетчик тиков таймера

    oldvect = _dos_getvect(0x1c); // Запоминаем адрес старого
                                // обработчика прерывания

    _dos_setvect(0x1c, timer);    // Устанавливаем свой
                                // обработчик

    printf("\nТаймер установлен. Нажмите любую"
           "клавишу...\n");

    getch(); // Ожидаем нажатия на любую клавишу

    _dos_setvect(0x1c,oldvect);    // Восстанавливаем старый
                                // обработчик прерывания
                                // таймера

    exit(0);
}

// Функция обрабатывает прерывания таймера
```



```
void _interrupt _far timer(void) {  
    ticks++; // Увеличиваем счетчик тиков таймера  
  
    // Если значение счетчика тиков кратно 20,  
    // выдаем сигнал на динамик компьютера  
    if((ticks % 20) == 0) BEEP();  
  
    // Вызываем старый обработчик прерывания  
    _chain_intr(oldvect);  
}
```

4.5. Особенности обработки аппаратных прерываний

Аппаратные прерывания вырабатываются устройствами компьютера, когда возникает необходимость их обслуживания. Например, по прерыванию таймера соответствующий обработчик прерывания увеличивает содержимое ячеек памяти, используемых для хранения времени. В отличие от программных прерываний, вызываемых запланировано самой прикладной программой, аппаратные прерывания всегда происходят асинхронно по отношению к выполняющимся программам. Кроме того, может возникнуть одновременно сразу несколько прерываний!

Для того чтобы система "не растерялась", решая какое прерывание обслуживать в первую очередь, существует специальная схема приоритетов. Каждому прерыванию назначается свой уникальный приоритет. Если происходит одновременно несколько прерываний, то система отдает предпочтение самому высокоприоритетному, откладывая на время обработку остальных прерываний.

Система приоритетов реализована на двух микросхемах Intel 8259 (для машин класса XT - на одной такой микросхеме). Каждая микросхема обслуживает до восьми приоритетов. Микросхемы можно объединять (каскадировать) для увеличения количества уровней приоритетов в системе.

Уровни приоритетов обозначаются сокращенно IRQ0 - IRQ15 (для машин класса XT существуют только уровни IRQ0 - IRQ7).

Для машин XT приоритеты линейно зависели от номера уровня прерывания. IRQ0 соответствовало самому высокому приоритету, за ним шли IRQ1, IRQ2, IRQ3 и т. д. Уровень IRQ2 в машинах класса XT был зарезервирован для дальнейшего расширения системы. И начиная с машин класса AT IRQ2 стал использоваться для каскадирования контроллеров прерывания 8259. Добавленные приоритетные уровни IRQ8 - IRQ15 в этих машинах располагаются по приоритету между IRQ1 и IRQ3.

На следующей странице приведена таблица аппаратных прерываний, расположенных в порядке приоритета, из которой видно, что самый высокий приоритет у прерываний от интервального таймера, затем идет прерывание от клавиатуры.

Номер	Описание
8	IRQ0 - прерывание интервального таймера, возникает 18,2 раза в секунду
9	IRQ1 - прерывание от клавиатуры. Генерируется при нажатии и при отжатии клавиши. Используется для чтения данных с клавиатуры
A	IRQ2 - используется для каскадирования аппаратных прерываний в машинах класса AT
70	IRQ8 - прерывание от часов реального времени
71	IRQ9 - прерывание от контроллера EGA
72	IRQ10 - зарезервировано
73	IRQ11 - зарезервировано
74	IRQ12 - зарезервировано
75	IRQ13 - прерывание от математического сопроцессора
76	IRQ14 - прерывание от контроллера жесткого диска
77	IRQ15 - зарезервировано
B	IRQ3 - прерывание асинхронного порта COM2
C	IRQ4 - прерывание асинхронного порта COM1
D	IRQ5 - прерывание от контроллера жесткого диска для XT
E	IRQ6 - прерывание генерируется контроллером флоппи-диска после завершения операции
F	IRQ7 - прерывание принтера. Генерируется принтером, когда он готов к выполнению очередной операции. Многие адаптеры принтера не используют это прерывание

Для управления схемами приоритетов необходимо знать внутреннее устройство контроллера прерываний 8259. Поступающие прерывания запоминаются в регистре запроса на прерывание IRR. Каждый бит из восьми в этом регистре соответствует прерыванию. После проверки на обработку в настоящий момент другого прерывания запрашивается информация из регистра обслуживания ISR. Перед выдачей запроса на прерывание в процессор проверяется содержимое восьмибитового регистра маски прерываний IMR. Если прерывание данного уровня не замаскировано, то выдается запрос на прерывание.

Наиболее интересными с точки зрения программирования контроллера прерываний являются регистры маски прерываний IMR и управляющий регистр прерываний.

В машинах класса XT регистр маски прерываний имеет адрес 21h, управляющий регистр прерываний - 20h. Для машин AT первый контроллер 8259 имеет

такие же адреса, что и в машинах XT, регистр маски прерываний второго контроллера имеет адрес A1h, управляющий регистр прерываний - A0h.

Разряды регистра маски прерываний соответствуют номерам IRQ. Для того чтобы замаскировать аппаратное прерывание какого-либо уровня, надо заслать в регистр маски байт, в котором бит, соответствующий этому уровню, установлен в 1. Например, для маскирования прерываний от НГМД в порт 21h надо заслать двоичное число 01000000.

Приведем пример программы, маскирующей прерывание от флоппи-диска:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void main(void);

void main(void) {
    outp(0x21, 0x40);
    printf("\nПрерывания от флоппи-диска запрещены.\n");
    exit(0);
}
```

Эта программа есть на дискете, прилагающейся к книге. Запустите ее (с жесткого диска) и попробуйте поработать, например, с дисководом A. У вас ничего не получится!

Чтобы "оживить" флоппи-диски, запустите программу, которая размаскирует все прерывания (в том числе и от флоппи):

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

void main(void);

void main(void) {
    outp(0x21, 0);
    printf("\nПрерывания от флоппи-диска разрешены.\n");
    exit(0);
}
```

Заметьте, что мы только что замаскировали прерывание именно от флоппи-диска, все остальные устройства продолжали нормально работать. Если бы мы выдали машинную команду CLI, то отключились бы все аппаратные прерывания. Это привело бы, например, к тому, что клавиатура была бы заблокирована.

Еще одно замечание, касающееся обработки аппаратных прерываний. Если вы полностью заменяете стандартный обработчик аппаратного прерывания, не забудьте в конце программы выдать байт 20h в порт с адресом 20h (A0h для второго контроллера 8259). Эти действия необходимы для очистки регистра обслу

живания прерывания ISR. При этом разрешается обработка прерываний с более низким приоритетом, чем то, которое только что обрабатывалось.

Если вы обрабатываете прерывание 1Ch, то добавка в конце программы не нужна, т. к. это прерывание является расширением другого прерывания (прерывания таймера).

Перед тем как завершить изучение прерываний, зададимся вопросом - можно ли замаскировать немаскируемое прерывание? Оказывается можно!

Конечно, если сигнал прерывания пришел на вход немаскируемого прерывания процессора, ничего сделать нельзя - прерывание произойдет неизбежно. Но в компьютерах XT и AT предусмотрены схемы, блокирующие вход немаскируемого прерывания процессора NMI.

Для XT маскированием немаскируемого прерывания управляет порт с адресом 0A0h. Если записать в него 0, немаскируемое прерывание будет запрещено, если 80h - разрешено.

Аналогично для AT маскированием немаскируемого прерывания управляет бит 7 порта 70h. Запись байта 0ADh в порт 70h запретит немаскируемое прерывание, а байта 2Dh - разрешит прохождение прерывания.

Заметим, что мы не запрещаем немаскируемое прерывание "внутри" процессора - это невозможно по определению, мы "не пускаем" сигнал прерывания на вход NMI.

Продолжение, начало см. стр. 35, 68

КУРС ПРАКТИЧЕСКОЙ РАБОТЫ С СИСТЕМОЙ АВТОКАД 10

С.А. Гладков, Ю.А. Кречко, К.И. Молодцов, В.В. Полищук, Г.А. Сучков

Оригинальное учебное пособие по курсу практической работы с пакетом Автокад, версия 10 (рассматриваются как английский, так и русифицированный варианты). Описаны команды системы Автокад, приемы работы с ними. Приводится описание языка программирования Автолисп и примеры его использования при модификации меню Автокада и для создания собственных команд.

Объем 13 л. Цена 35 руб.

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ АВТОЛИСП В СИСТЕМЕ САПР АВТОКАД

С.А. Гладков

Учебно-справочное пособие. Описывается структура языка, распределение памяти и обращение к графической базе данных Автокада, дается методика написания и примеры программ. Рассмотрены вопросы написания макроопределений, меню Автокада и настройки рабочей среды. Содержит предметный указатель функций Автолиспа. Ориентировано на пользователей персональных компьютеров, работающих в области САПР.

Объем 5 л. Цена 19 руб.

Продолжение на стр. 220, 236

РЕЗИДЕНТНЫЕ ПРОГРАММЫ

5.1. Что такое резидентная программа?

Обычно после завершения очередной выполняющейся программы DOS освобождает место в памяти, которое занимала программа, чтобы загрузить на это место новую. Однако есть способ оставить программу в памяти и после ее завершения. Следующая запускаемая программа при этом разместится в памяти после оставленной.

Остающиеся в памяти после завершения своей работы программы называются резидентными или TSR (Terminate and Stay Resident).

Для чего может понадобиться составление TSR-программ? Существует несколько приложений, для которых подходят TSR-программы. Чаще всего TSR-программы используются для русификации импортных персональных компьютеров. Написано множество программ для загрузки русских шрифтов в память видеоадаптеров, для печати русских букв на принтере в графическом режиме, для русификации клавиатуры и т. п. Для всех этих программ характерно то, что они запускаются один раз при загрузке компьютера - их имена обычно включают в AUTOEXEC.BAT. Эти программы могут переключать на себя обработку прерываний, связанных с выводом на печать или с обращением к клавиатуре и/или выполнять разовые инициализирующие действия, такие, как загрузка русских шрифтов в память видеоадаптера.

Другим примером использования TSR-программ являются программы резидентных калькуляторов, справочных баз данных типа Norton Guide или целых интегрированных систем наподобие SideKick фирмы Borland. Такие программы тоже обычно запускаются через AUTOEXEC.BAT или при необходимости. Они перехватывают клавиатурные прерывания и отслеживают нажатие клавиш. Как только обнаруживается нажатие определенной заранее клавиши, поверх имеющегося на экране изображения выводится окно диалога резидентной программы.

Иногда TSR-программы используют для обслуживания нестандартной аппаратуры, особенно когда с этой аппаратурой должны работать по очереди несколько разных программ. В этом случае TSR-программа встраивает обработчик какого-либо прерывания, не занятого системой, и через этот обработчик все прикладные программы обращаются к нестандартной аппаратуре.

Аналогично работает резидентная часть системы управления базами данных ORACLE. Через прерывание, устанавливаемое при запуске ORACLE, прикладная программа, составленная на любом языке программирования, имеющем средство вызова прерывания, может пользоваться базой данных ORACLE.

Для обслуживания нестандартной аппаратуры больше подходят загружаемые драйверы устройств, о которых мы еще будем подробно говорить. Драйверы предоставляют намного более гибкие и богатые средства общения с устройствами,

чем TSR-программы. Однако если все, что вы собираетесь сделать, - это обработать несколько прерываний или обслужить какое-нибудь несложное устройство, то TSR-программы - это то, что вам нужно.

TSR-программы не могут свободно пользоваться прерываниями DOS, поскольку программы DOS (и BIOS) не обладают свойством реентерабельности (повторной входимости). Об этом мы будем говорить при обсуждении особенностей резидентных программ.

5.2. Как программе стать резидентной

У вас есть две возможности оставить свою программу резидентной в памяти - использовать прерывание INT 27h или функцию 31h прерывания INT 21h.

Для использования прерывания 27h сегментный регистр CS должен указывать на PSP-программы, а в регистре DX должно быть записано смещение последнего байта программы плюс один байт. Нетрудно заметить, что этот способ остаться резидентной больше всего подходит для программ в формате COM. Вы не сможете оставить резидентной программу длиннее 64 Кбайт.

Другой, более удобный способ - использовать функцию 31h прерывания INT 21h. В регистре AL вы можете указать код завершения программы, регистр DX в этом случае должен содержать длину резидентной части программы в параграфах. Здесь уже нет ограничения 64 Кбайт на длину программы. Использование этой функции - единственная возможность оставить резидентной программу длиннее 64 Кбайт.

Но не стоит увлекаться длинными TSR-программами, т. к. обычно освободить память, занимаемую ставшей уже ненужной резидентной программой, можно только с помощью перезагрузки операционной системы.

Библиотека функций QuickC содержит специальную функцию для оставления программы резидентной в памяти. Эта функция использует прерывание INT 21h (функция 31h) и имеет имя `_dos_keep()`. Первый параметр функции - код завершения (то, что записывается в регистр AL), а второй - длина резидентной части программы в параграфах.

Нужно внимательно следить за размером оставляемой резидентной части программы, не забывать про области данных и буфера. Если вы укажете слишком малый объем памяти для TSR-программы, то прикладная программа, загруженная сразу за ней, может оказаться разрушенной.

5.3. Вызов резидентной программы

Возможны два способа вызова резидентной программы - либо прикладная программа выдает прерывание, обрабатываемое TSR-программой, либо сама TSR-программа отслеживает нажатие клавиш оператором компьютера и в случае нажатия определенной клавиши (или комбинации клавиш) запускает диалоговую часть резидентной программы.

Первый способ используется прикладными программами, работающими с нестандартной аппаратурой, второй - диалоговыми резидентными программами.

Самый простой способ организовать связь оператора с TSR-программой - это использовать прерывание по нажатию клавиши печати содержимого экрана PrtScr. Но при этом вы теряете возможность распечатки содержимого экрана. Впрочем, иногда вам и нужно, чтобы вместо печати содержимое экрана записывалось, например, в файл. Или вы можете составить свою собственную программу печати содержимого экрана (используя графическую печать или какие-либо особенности принтеров).

Если вы используете прерывание 9, вырабатываемое при каждом нажатии на клавишу, для определения момента запуска диалога, не следует забывать об организации цепочки прерываний, с тем чтобы после анализа (или перед анализом) передать управление стандартному обработчику 9-го прерывания.

Подробно о работе с клавиатурой и о 9-м прерывании будет рассказано во втором томе книги.

5.4. Особенности резидентных программ

TSR-программы имеют некоторые особенности, отличающие их от "нормальных" программ.

Им не разрешается использовать DOS-прерывания, когда вздумается. Это связано с тем, что DOS проектировалась как однозадачная операционная система, поэтому модули DOS не обладают свойством реентерабельности. Что это означает на практике?

Допустим, ваша программа записывает длинный файл на диск. Во время записи вы (возможно, случайно) нажали клавишу, активизирующую TSR-программу записи содержимого экрана в файл.

Теперь доступа к диску требуют две программы - прикладная, записывающая длинный файл, и ваша TSR-программа. Запись файла из прикладной программы приостановится, далее произойдет запись копии экрана в файл, после чего возобновится запись файла из прикладной программы. Все было бы хорошо, если бы прикладная программа и TSR-программа не использовали одни и те же внутренние области данных DOS для работы с диском. При запуске TSR-программа безвозвратно испортит текущее состояние служебных областей данных, которые прикладная программа использовала при записи на диск.

И таких примеров можно привести много. BIOS также далеко не весь реентерабельный. TSR-программа может смело использовать разве лишь прерывание 16h для работы с клавиатурой, которое реентерабельно. Для вывода на экран лучше всего использовать непосредственную запись символов в видеопамять дисплейного адаптера.

Не стоит пользоваться многими функциями библиотеки QuickC, т. к. они могут вызывать прерывания DOS. Например, функция malloc() вызывает прерывание DOS для определения размера свободной памяти в системе.

Возможны трудности с выполнением арифметических действий с числами в формате с плавающей запятой, т. к. функция _dos_keep() при завершении прог-

раммы восстанавливает прерывания, использовавшиеся для эмуляции арифметики с плавающей запятой и для работы с арифметическим сопроцессором.

Некоторые из перечисленных проблем (те, что связаны с использованием прерываний DOS) можно решить с помощью недокументированного прерывания INT 28h.

Это прерывание вызывается DOS при ожидании ввода с клавиатуры. В этот момент вы можете использовать любое прерывание DOS, кроме функций от 00h до 0Ch прерывания INT 21h. Утилита спулинга печати PRINT.COM использует это прерывание.

Можно рекомендовать следующий способ использования прерывания 28h. Обработчик прерывания 9 отслеживает нажатие клавиши, которая должна активизировать TSR-программу. Обнаружив эту клавишу (или комбинацию клавиш), обработчик прерывания 9 устанавливает флаг запроса на активизацию TSR-программы и завершает свою работу обычным способом.

Ваша TSR-программа должна создать свой обработчик прерывания 28h и сцепить его со стандартным. Каждый раз, когда DOS ожидает ввода с клавиатуры (в этот момент DOS не использует свои прерывания), вызывается прерывание 28h. Ваш обработчик проверяет флаг активизации, устанавливаемый обработчиком прерывания 9, и, если флаг установлен, TSR-программа активизируется и может пользоваться услугами DOS, в частности файловой системой.

Разумеется, после выполнения всех необходимых действий TSR-программа должна сбросить флаг активизации.

Можно также вместе с прерыванием 28 использовать аппаратное прерывание таймера с номером 8. В этом случае надо проверять не только флаг активизации, но и так называемый флаг критической секции DOS. Это байт, адрес которого возвращает недокументированная функция 34h прерывания DOS 21h в регистрах ES:BX. Если этот байт равен 0, то DOS не использует свои прерывания и наступил подходящий момент для активизации TSR-программы.

TSR-программа может вступить в конфликт с другими TSR-программами или прикладным обеспечением, если будет использовать занятые ими номера прерываний. Известен случай, когда драйвер клавиатуры и экрана (SDRIVER) вызывал зависания программных продуктов фирмы Microsoft из-за того, что эти продукты выдавали 16h прерывание с содержимым регистра AX, равным 5500h, 5501h и т. д. Обработчик 16h-го прерывания программы SDRIVER при этом заклинивался из-за ошибки в программе (или, точнее, из-за плохой реакции на такого рода номера функций прерывания 16h).

Очень важно не допустить случайного повторного запуска TSR-программы, т. к. повторное переназначение векторов прерываний почти наверняка приведет систему к краху.

Для проверки наличия TSR-программы в памяти обычно используют прерывание мультиплексора 2Fh, специально предназначенное для организации элементов мультизадачности в DOS. Это прерывание используется спулером печати PRINT.COM (об этом мы будем говорить при описании работы с принтером).

TSR-программа может переназначить это прерывание на себя и сделать так, чтобы оно "откликлось" на какой-либо код функции, зарезервированный для прикладных программ. Можно использовать коды C0h...FFh. При запуске TSR-программа вызывает прерывание 2Fh с выбранным кодом и проверяет ответ (передаваемый, например, в регистре AX). Если прерывание отвечает тем кодом, который задан в TSR-программе, это означает, что копия программы уже есть в памяти и повторная установка недопустима.

Приведенные ниже примеры проиллюстрируют все эти особенности.

5.5. Примеры резидентных программ

Приведем несколько примеров TSR-программ.

Первая программа перехватывает прерывание 9 (аппаратное прерывание клавиатуры). Запустив эту программу из приглашения DOS, вы сможете убедиться в том, что прерывание от клавиатуры возникает не только тогда, когда вы нажимаете на клавишу, но и когда ее отпускаете.

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>

// Выключаем проверку стека и указателей

#pragma check_stack( off )
#pragma check_pointer( off )

// Макро для подачи звукового сигнала
#define BEEP() _asm { \
                        _asm xor    bx, bx      \
                        _asm mov    ax, 0E07h    \
                        _asm int    10h          \
                    }

// Указатель на старую функцию обработки 9-го прерывания
void (_interrupt _far *oldkey)( void );

// Объявление новой функции обработки 9-го прерывания
void _interrupt _far newkey( void );

void main(void);

void main(void) {
    unsigned size; // Размер резидентной части TSR-программы
    char _far *newstack; // Указатель на новый стек,
                        // который будет использовать
                        // TSR-программа
```



```

char _far *tsrbottom; // Указатель на конец
                        // TSR-программы, используется
                        // для определения размера
                        // резидентной части

// Записываем адрес стека TSR-программы

_asm mov  WORD PTR newstack[0], sp
_asm mov  WORD PTR newstack[2], ss

FP_SEG(tsrbottom) = _psp; // Указатель конца
FP_OFF(tsrbottom) = 0;    // программы устанавливаем
                        // на начало PSP

// Вычисляем размер программы в параграфах
// Добавляем 1 параграф на случай
// некротной параграфу длины
size = ((newstack - tsrbottom) >> 4) + 1;

// Встраиваем свой обработчик прерывания 9,
// запоминаем старый вектор прерывания 9

oldkey = _dos_getvect(0x9);
_dos_setvect(0x9, newkey);

// Завершаем программу и остаемся в памяти
_dos_keep(0, size);
}

// Новый обработчик клавиатурного прерывания
void _interrupt _far newkey() {
    BEEP(); // Выдаем звуковой сигнал

    // Вызываем стандартный обработчик прерывания 9
    _chain_intr( oldkey );
}

```

Следующая программа GRAB демонстрирует использование функций DOS в TSR-программах. Она содержит все элементы, необходимые "безопасным" резидентным программам.

Программа предназначена для копирования содержимого видеобуфера в файл. Запись в файл активизируется при нажатии комбинации клавиш CTRL+PrtSc. После каждой записи имя файла изменяется.

В самом начале своей работы программа проверяет наличие своей копии в памяти, т. к. повторное переназначение векторов прерываний приведет систему к краху. Некоторые тонкости, связанные с программированием клавиатуры и видеоадаптера, с получением адреса видеобуфера используются в программе без

объяснения. Вся необходимая информация будет приведена позже, в главах, посвященных программированию клавиатуры и видеоадаптеров.

Итак, текст программы:

```
#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
#include "sysp.h"

// Выключаем проверку стека и указателей
#pragma check_stack( off )
#pragma check_pointer( off )

// Макро для подачи звукового сигнала
#define BEEP() _asm { \
                    _asm xor    bx, bx      \
                    _asm mov    ax, 0E07h    \
                    _asm int    10h          \
                }

// Указатели на старые обработчики прерываний
void (_interrupt _far *old8)(void); // Таймер
void (_interrupt _far *old9)(void); // Клавиатура
void (_interrupt _far *old28)(void); // Занятость DOS
void (_interrupt _far *old2f)(void); // Мультиплексор

// Новые обработчики прерываний
void _interrupt _far new8(void);
void _interrupt _far new9(void);
void _interrupt _far new28(void);
void _interrupt _far new2f(unsigned _es, unsigned _ds,
                           unsigned _di, unsigned _si,
                           unsigned _bp, unsigned _sp,
                           unsigned _bx, unsigned _dx,
                           unsigned _cx, unsigned _ax,
                           unsigned _ip, unsigned _cs,
                           unsigned flags);

int iniflag; // Флаг запроса на вывод экрана в файл
int outflag; // Флаг начала вывода в файл
int name_counter; // Номер текущего выводимого файла
char _far *crit; // Адрес флага критической секции DOS

// =====

void main(void);
void main(void) {

    union REGS inregs, outregs;
    struct SREGS segregs;

    unsigned size; // Размер резидентной части TSR-программы
```



```
// Вызываем прерывание мультиплексора с AX = FF00
// Если программа GRAB уже запускалась, то новый
// обработчик прерывания мультиплексора вернет
// в регистре AX значение 00FF.
// Таким способом мы избегаем повторного изменения
// содержимого векторной таблицы прерываний.

inregs.x.ax = 0xff00;
int86(0x2f, &inregs, &outregs);

if(outregs.x.ax == 0x00ff) {
    printf("\nПрограмма GRAB уже загружена\n");
    hello();
    exit(-1);
}

// Выдаем инструкцию по работе с программой GRAB
hello();

// Вычисляем размер программы в параграфах
// Добавляем 1 параграф на случай
// некротной параграфу длины

size = (12000 >> 4) + 1;

// Устанавливаем начальные значения флагов

outflag=iniflag=0;

// Сбрасываем счетчик файлов. Первый файл будет
// иметь имя GRAB0.DOC. В дальнейшем этот счетчик
// будет увеличивать свое значение на 1.

name_counter=0;

// Получаем указатель на флаг критической секции DOS.
// Когда этот флаг равен 0, TSR-программа может
// пользоваться функциями DOS

inregs.h.ah = 0x34;
intdosx( &inregs, &outregs, &segregs );
crit=(char __far *)FP_MAKE(segregs.es,outregs.x.bx);

// Устанавливаем собственные обработчики прерываний.

old9 = _dos_getvect(0x9);
_dos_setvect(0x9, new9);

old8 = _dos_getvect(0x8);
_dos_setvect(0x8, new8);

old28 = _dos_getvect(0x28);
_dos_setvect(0x28, new28);

old2f = _dos_getvect(0x2f);
_dos_setvect(0x2f, new2f);
```



```
// Завершаем программу и остаемся в памяти
_dos_keep(0, size);
}

// =====

// Новый обработчик прерывания мультиплексора.
// Используется для предохранения программы
// от повторного встраивания в систему как резидентной.

void _interrupt _far new2f(unsigned _es, unsigned _ds,
                           unsigned _di, unsigned _si,
                           unsigned _bp, unsigned _sp,
                           unsigned _bx, unsigned _dx,
                           unsigned _cx, unsigned _ax,
                           unsigned _ip, unsigned _cs,
                           unsigned flags)
{
    // Если прерывание вызвано с содержимым регистра AX,
    // равным FF00, возвращаем в регистре AX значение
    // 00FF, в противном случае передаем управление
    // старому обработчику прерывания

    if(_ax != 0xff00) _chain_intr(old2f);
    else _ax = 0x00ff;
}

// =====

// Новый обработчик аппаратного прерывания таймера

void _interrupt _far new3(void) {
    // Вызываем старый обработчик
    (*old8)();

    // Если была нажата комбинация клавиш CTRL+PrtSc
    // (iniflag при этом устанавливается в 1
    // новым обработчиком прерывания 9) и
    // если запись в файл уже не началась,
    // то при значении флага критической секции
    // DOS, равном 0, выводим содержимое экрана в файл

    if((iniflag != 0) && (outflag == 0) && *crit == 0) {
        outflag=1; // Устанавливаем флаг начала вывода
        _enable(); // Разрешаем прерывания

        write_buf(); // Записываем содержимое
                     // буфера экрана в файл

        outflag=0; // Сбрасываем флаги в исходное
        iniflag=0; // состояние
    }
}
```



```
// =====  
  
// Новый обработчик прерывания 28h, которое вызывает DOS,  
// если она ожидает ввода от клавиатуры. В этот момент  
// TSR-программа может пользоваться функциями DOS.  
  
void _interrupt _far new28(void) {  
    // Если была нажата комбинация клавиш CTRL+PrtSc  
    // (iniflag при этом устанавливается в 1 новым  
    // обработчиком прерывания 9) и если уже не началась  
    // запись в файл, то выводим содержимое экрана в файл  
  
    if((iniflag != 0) && (cutflag == 0)) {  
        outflag=1; // Устанавливаем флаг начала вывода  
        _enable(); // Разрешаем прерывания  
  
        write_buf(); // Записываем содержимое видеобуфера  
                     // в файл  
  
        outflag=0;   // Сбрасываем флаги в исходное  
        iniflag=0;   // состояние  
    }  
  
    // Передаем управление старому обработчику прерывания 28  
    _chain_intr(old28);  
}  
  
// =====  
  
// Новый обработчик клавиатурного прерывания. Он фиксирует  
// нажатие комбинации клавиш CTRL+PrtSc и устанавливает флаг  
// iniflag, который сигнализирует о необходимости выбрать под-  
// ходящий момент и записать содержимое видеобуфера в файл  
  
void _interrupt _far new9(void) {  
    // Если SCAN-код равен 0x37 (клавиша PrtSc),  
    // нажата клавиша CTRL (бит 4 байта состояния клавиатуры,  
    // находящийся в области данных BIOS по адресу 0040:0017  
    // установлен в 1) и если не установлен флаг iniflag,  
    // то устанавливаем флаг iniflag в 1.  
  
    if((inp(0x60) == 0x37) && (iniflag == 0) &&  
        (*(char _far *)FP_MAKE(0x40,0x17) & 4) != 0) {  
        // Выдаем звуковой сигнал  
  
        BEEP();  
        BEEP();  
        BEEP();  
  
        _disable(); // Запрещаем прерывания
```



```
// Разблокируем клавиатуру и разрешаем прерывания
_asm {
    in  al,61h
    mov ah,al
    or  al,80h
    out 61h,al
    xchg ah,al
    out 61h,al

    mov al,20h
    out 20h,al
}

// Устанавливаем флаг запроса на запись содержимого
// видеобуфера в файл
iniflag = 1;
_enable(); // Разрешаем прерывания
}

// Если нажали не CTRL+PrtSc, то передаем управление старому
// обработчику прерывания 9
else _chain_intr(old9);
}

// =====
// Функция возвращает номер текущего видеорежима
int get_vmode(void) {
    char _far *ptr;

    ptr = FP_MAKE(0x40,0x49); // Указатель на байт
                             // текущего видеорежима
    return(*ptr);
}

// =====
// Функция возвращает сегментный адрес видеобуфера.
// Учитывается содержимое регистров смещения адреса видеобуфера
int get_vbuf(int vmode) {
    unsigned vbase;
    unsigned adr_6845;
    unsigned high;
    unsigned low;
    unsigned offs;

    // В зависимости от видеорежима базовый адрес
    // видеобуфера может быть 0xb000 или 0xb800
    vbase = (vmode == 7) ? 0xb000 : 0xb800;
```



```
// получаем адрес порта видеоконтроллера 6845
adr_6845 = *(unsigned _far *)(FP_MAKE(0x40,0x63));
// Считываем содержимое регистров 12 и 13 видеоконтроллера
outp(adr_6845,0xc);
high = inp(adr_6845+1);
outp(adr_6845,0xd);
low = inp(adr_6845+1);
offs = ((high << 8) + low) >> 4;
// Добавляем к базовому адресу видеобuffers
// смещение, взятое из регистров видеоконтроллера
vbase += offs;
return(vbase);
}

// =====

// Функция возвращает количество символов в строке
// для текущего видеорежима
int get_column(void) {
    return(*(int _far *)(FP_MAKE(0x40,0x4a)));
}

// =====

// Функция возвращает количество строк
// для текущего видеорежима
int get_row(void) {
    unsigned char ega_info;
    ega_info = *(unsigned char _far *)(FP_MAKE(0x40,0x87));
    // Если нет EGA, то используется 25 строк, если EGA
    // присутствует, считываем число строк. Это число
    // находится в области данных BIOS по адресу 0040:0084.
    if(ega_info == 0 || ((ega_info & 8) != 0) ) {
        return(25);
    }
    else {
        return(*(unsigned char _far *)
            (FP_MAKE(0x40,0x84)) + 1);
    }
}
```



```
// =====  
// Функция записи содержимого видеобуфера в файл  
int write_buf(void) {  
    // Видеопамять состоит из байтов символов и байтов  
    // атрибутов. Нам нужны байты символов chr.  
    typedef struct _VIDEOBUF_  
    {  
        unsigned char chr;  
        unsigned char attr;  
    } VIDEOBUF;  
  
    VIDEOBUF _far *vbuf;  
    int i, j, k, max_col, max_row;  
    FILE *out_file;  
    char fname[20], ext[8];  
  
    i=get_vmode(); // Получаем номер текущего видеорежима  
    // Для графического режима ничего не записываем  
    if(i > 3 && i != 7) return(-1);  
    // Устанавливаем указатель vbuf на видеобуфер  
    vbuf=(VIDEOBUF _far *)FP_MAKE(get_vbuf(i),0);  
    // Определяем размеры экрана  
    max_col = get_column();  
    max_row = get_row();  
    // Формируем имя файла для записи образа экрана  
    itoa(name_counter++.ext,10);  
    strcpy(fname,"!grab");  
    strcat(fname,ext);  
    strcat(fname,".doc");  
    out_file=fopen(fname,"wb+");  
    // Записываем содержимое видеобуфера в файл  
    for(i=0; i<max_row; i++) {  
        for(j=0; j<max_col; j++) {  
            fputc(vbuf->chr,out_file);  
            vbuf++;  
        }  
  
        // В конце каждой строки добавляем символы  
        // перевода строки и возврата каретки  
        fputc(0xd,out_file);  
        fputc(0xa,out_file);  
    }  
}
```



```

        fclose(out_file);
        return(0);
    }

// =====

// Функция выводит на экран инструкцию по
// использованию программы GRAB

int hello(void) {
    printf("\nУтилита копирования содержимого"
           "\nэкрана в файл GRAB<n>.DOC"
           "\nCopyright (C)Frolov A.,1990"
           "\n"
           "\nДля копирования нажмите CTRL+PrtSc"
           "\n"),
}

```

Приведем пример TSR-программы, написанной на языке Ассемблера. Эта программа переназначает прерывание 13h, которое используется для работы с дисками. Она позволяет организовать защиту диска от записи. При первом запуске программа включает защиту, при втором выключает, потом опять включает и т. д. В качестве флага - признака включения или выключения защиты - используется компонента смещения вектора прерывания F0h, зарезервированного для интерпретатора BASIC.

```

        .MODEL tiny
        .CODE
        .STARTUP

        jmp begin

old_int13h_off  dw 0    ; Адрес старого обработчика
old_int13h_seg  dw 0    ; прерывания 13h

old_int2Fh_off  dw 0    ; Адрес старого обработчика
old_int2Fh_seg  dw 0    ; прерывания 2Fh

; Новый обработчик прерывания 2Fh нужен для проверки наличия
; программы в памяти при ее запуске для предохранения
; от повторного запуска

new_int2Fh  proc  far
            cmp     ax,0FF00h
            jz      installed

            jmp     dword ptr cs:old_int2Fh_off

; Если код функции 0FF00h, то возвращаем в регистре AX значение
; 00FFh. Это признак того, что программа уже загружена в память

installed:
            mov     ax,00FFh
            iret

new_int2Fh  endp

```


; Новый обработчик прерывания 13h. Для команд записи
; на жесткий диск выполняет проверку содержимого
; компоненты смещения вектора прерывания FFh.
; Эта ячейка служит для триггерного переключения режима
; работы прерывания 13h – включения/выключения защиты записи.

new_int13h proc far

cmp ah,3 ; запись сектора
je protect
cmp ah,5 ; форматирование трека
je protect
jmp dword ptr cs:old_int13h_off

old_int13h:

pop es
pop bx
pop ax

jmp dword ptr cs:old_int13h_off

protect:

push ax
push bx
push es

; Проверяем значение триггерного флага защиты

xor ax,ax
mov es,ax
mov bx,0F0h*4
mov ax,WORD PTR es:[bx]
cmp ax,0FFFFh

jne old_int13h

; Для флоппи-дисков защиту не включаем

cmp dl,0
je old_int13h
cmp dl,1
je old_int13h

pop es
pop bx
pop ax

; Имитируем ошибку записи при попытке
; записать данные на защищенный от записи диск

mov ah,3
stc
ret 2

new_int13h endp


```
;=====
; Точка входа в программу
begin proc far

; Проверяем, не загружена ли уже программа в память
    mov     ax,0FF00h
    int     2Fh

    cmp     ax,00FFh
    jne     first_start

    jmp     invert_protect_flag

; Первоначальный запуск программы
first_start:

; Устанавливаем триггерный флаг защиты записи в состояние,
; соответствующее включенной защите
    xor     ax,ax
    mov     es,ax
    mov     bx,0F0h*4
    mov     WORD PTR es:[bx],0FFFFh

; Запоминаем адрес старого обработчика прерывания 13h
    mov     ax,3513h
    int     21h
    mov     cs:old_int13h_off,bx
    mov     cs:old_int13h_seg,es

; Запоминаем адрес старого обработчика прерывания 2Fh
    mov     ax,352Fh
    int     21h
    mov     cs:old_int2Fh_off,bx
    mov     cs:old_int2Fh_seg,es

    push    cs
    pop     ds

; Выводим сообщение о включении защиты
    mov     dx,offset msg_on
    mov     ah,9
    int     21h

; Устанавливаем новые обработчики прерываний 13h и 2Fh
    mov     dx,OFFSET new_int13h
    mov     ax,2513h
    int     21h
```



```
    mov     dx,OFFSET new_int2Fh
    mov     ax,252Fh
    int     21h

; Завершаем программу и оставляем резидентно в памяти часть
; программы, содержащую новые обработчики прерываний

    mov     dx,OFFSET begin
    int     27h

; Если это не первый запуск программы,
; инвертируем содержимое триггерного флага защиты
invert_protect_flag:

    xor     ax,ax
    mov     es,ax
    mov     bx,0F0h*4

    mov     ax,WORD PTR es:[bx]
    not     ax
    mov     WORD PTR es:[bx],ax
    mov     cx,ax

    cmp     cx,0FFFFh
    je      prot_on

; Выводим сообщение с выключении защиты

    mov     dx,OFFSET msg_off
    jmp     short send_msg

prot_on:

; Выводим сообщение о включении защиты

    mov     dx,OFFSET msg_on
send_msg:
    mov     ah,9
    push    cs
    pop     ds
    int     21h

    .EXIT

begin endp

msg_on      db 'Защита диска включена$'
msg_off     db 'Защита диска ВЫКЛЮЧЕНА$'

end
```

Этим примером мы завершим обзор TSR-программ. В следующей главе будет описан другой вид резидентных программ - драйверы. Использование драйвера - более предпочтительный способ организации обслуживания нестандартной аппаратуры, чем TSR-программы.

ПРИЛОЖЕНИЕ 1

УПРАВЛЯЮЩИЕ БЛОКИ

Векторная таблица связи MS-DOS:

(-2)	2	mcb_seg	- сегмент первого управляющего блока памяти (MCB)
(0)	4	dev_cb	- указатель на первый блок управления устройствами DOS (DOS Device Control Block)
(+4)	4	file_tab	- указатель на таблицу файлов DOS
(+8)	4	clock_dr	- указатель на драйвер CLOCK\$, установленный или резидентный
(+12)	4	con_dr	- указатель на актуальный драйвер CON, установленный или резидентный

----- DOS 2.x -----

(+16)	1	num_lgdr	- число логических драйверов в системе
(+17)	2	max_btbl	- максимальное число байт/блоков любого блочного устройства
(+19)	4	disk_buf	- указатель на первый дисковый буфер
(+23)	-	null_dr	- начало драйвера NUL - первого драйвера в списке драйверов DOS

----- DOS 3.x, 4.x -----

(+16)	2	max_btbl	- максимальное число байт в блоке блочного устройства
(+18)	4	disk_buf	- указатель на первый дисковый буфер
(+22)	4	drv_info	- указатель на массив информации об устройствах
(+26)	4	fcbl_tabl	- указатель на таблицу FCB
(+30)	2	fcbl_size	- размер таблицы FCB
(+32)	1	num_bdev	- число блочных устройств
(+33)	1	lastdriv	- значение LASTDRIVE в файле CONFIG.SYS (по умолчанию равно 5)
(+34)	-	null_dr	- начало драйвера NUL - первого драйвера в списке драйверов DOS

Блок управления памятью MCB:

(0)	1	type	- тип блока MCB (M или Z)
(+1)	2	owner	- параграф владельца блока (если 0, то блок описывает сам себя)

(+3)	2	size	- число параграфов в этом блоке (один параграф имеет размер 16 байт)
(+5)	11	reserve	- зарезервировано

Префикс программного сегмента PSP:

(0)	2	int20h	- двоичный код команды int 20h (программы могут использовать эту команду для завершения своей работы)
(+2)	2	mem_top	- нижняя граница доступной памяти в системе в параграфах
(+4)	1	reserv1	- зарезервировано
(+5)	5	call_dsp	- команда вызова FAR CALL диспетчера MS-DOS
(+10)	4	term_adr	- адрес завершения (Terminate Address)
(+14)	4	cbrk_adr	- адрес обработчика CTRL+BREAK
(+18)	4	crit_err	- адрес обработчика критической ошибки
(+22)	2	parn_psp	- сегмент PSP программы, запустившей данную программу (программы-родителя)
(+24)	20	file_tab	- таблица открытых файлов, если здесь находятся байты 0FFh, то таблица не используется
(+44)	2	env_seg	- сегмент блока памяти, содержащего переменные среды
(+46)	4	ss_sp	- адрес стека SS:SP программы
(+50)	2	max_open	- максимальное число открытых файлов
(+52)	4	file_tba	- адрес таблицы открытых файлов
(+56)	24	reserv2	- зарезервировано
(+80)	3	disp	- диспетчер функций DOS
(+83)		reserv3	- зарезервировано
(+92)	16	fcbl	- форматируется как стандартный FCB, если первый аргумент командной строки содержит правильное имя файла
(+108)	20	fcbl	- заполняется для второго аргумента командной строки аналогично fcbl
(+128)	1	p_size	- число значащих символов в неформатированной области параметров либо буфер обмена с диском DTA, назначенный по умолчанию
(+129)	127	parm	- неформатированная область параметров, заполняется при запуске программы из командной строки

Формат блока DDCB для MS-DOS версий 2.x и 3.x:

(0)	1	drv_num	- номер устройства (0 соответствует устройству A, 1 - B и т. д.)
(+1)	1	drv_numd	- дополнительный номер устройства внутри драйвера
(+2)	2	sec_size	- размер сектора в байтах
(+4)	1	clu_size	- число, на единицу меньшее количества секторов в кластере
(+5)	1	clu_base	- число, являющееся степенью 2 числа секторов в кластере
(+6)	2	boot_siz	- количество зарезервированных секторов (boot-секторы, начало корневого каталога)
(+8)	1	fat_num	- количество копий FAT
(+9)	2	max_dir	- максимальное число дескрипторов файлов в корневом каталоге (т. е. максимальное число файлов, которое может содержать корневой каталог на этом устройстве)
(+11)	2	data_sec	- номер первого сектора данных на диске (номер сектора, соответствующего кластеру номер 2)
(+13)	2	hi_clust	- максимальное количество кластеров (равно увеличенному на 1 количеству кластеров данных)
(+15)	1	fat_size	- количество секторов, занимаемых копией FAT
(+16)	2	root_sec	- номер первого сектора корневого каталога
(+18)	4	drv_addr	- FAR-адрес заголовка драйвера, обслуживающего данное устройство
(+22)	1	media	- байт описания среды носителя данных
(+23)	1	acc_flag	- флаг доступа, 0 означает, что к устройству был доступ
(+24)	4	next	- адрес следующего блока DDCB, для последнего блока в поле смещения находится число FFFF
----- только для DOS 2.x -----			
(+28)	2	dir_clu	- номер начального кластера текущего каталога (0 для корневого каталога)
(+30)	64	dir_path	- строка в формате ASCIIZ, содержащая путь к текущему каталогу
----- DOS 3.x -----			
(+28)	2	reserv1	- зарезервировано, обычно равно 0
(+30)	2	built	- число FFFF в этом поле означает, что блок DDCB был построен

Формат блока DDCB для MS-DOS версии 4.x :

(0)	1	drv_num	- номер устройства (0 соответствует устройству А, 1 - В и т. д.)
(+1)	1	drv_numd	- дополнительный номер устройства внутри драйвера
(+2)	2	sec_size	- размер сектора в байтах
(+4)	1	clu_size	- число, на единицу меньшее количества секторов в кластере
(+5)	1	clu_base	- число, являющееся степенью 2 числа секторов в кластере
(+6)	2	boot_siz	- количество зарезервированных секторов (boot-секторы, начало корневого каталога)
(+8)	1	fat_num	- количество копий FAT
(+9)	2	max_dir	- максимальное число дескрипторов файлов в корневом каталоге (т. е. максимальное число файлов, которое может содержать корневой каталог на этом устройстве)
(+11)	2	data_sec	- номер первого сектора данных на диске (номер сектора, соответствующего кластеру номер 2)
(+13)	2	hi_clust	- максимальное количество кластеров (равно увеличенному на 1 количеству кластеров данных)
(+15)	1	fat_size	- количество секторов, занимаемых копией FAT
(+16)	1	reserv1	- зарезервировано
(+17)	2	root_sec	- номер первого сектора корневого каталога
(+19)	4	drv_addr	- FAR-адрес заголовка драйвера, обслуживающего данное устройство
(+23)	1	media	- байт описания среды носителя данных
(+24)	1	acc_flag	- флаг доступа, 0 означает, что к устройству был доступ
(+25)	4	next	- адрес следующего блока DDCB, для последнего блока в поле смещения находится число FFFF
(+29)	2	reserv2	- зарезервировано
(+31)	2	built	- число FFFF в этом поле означает, что блок DDCB был построен

Формат таблицы файлов для DOS 3.x:

(0)	4	next	- указатель на следующую таблицу файлов
(+4)	2	file_count	- количество файлов в этой таблице
---- Дальше идут блоки DFCB в количестве file_count штук ----			
(0)	2	handl_num	- количество файловых чисел, связанных с данным файлом (file handle)
(+2)	1	access_mode	- режим доступа к файлу, заданный при открытии файла
(+3)	2	reserv1	- зарезервировано
(+5)	2	dev_info	- информация IOCTL, полученная для устройства, на котором расположен этот файл (подробно формат и назначение этого поля будут рассмотрены в главе, посвященной драйверам)
(+7)	4	driver	- указатель на драйвер, обслуживающий устройство, содержащее файл
(+11)	2	first_clu	- номер первого кластера, распределенного файлу
(+13)	2	time	- время последнего изменения файла в упакованном формате
(+15)	2	date	- дата последнего изменения файла в упакованном формате
(+17)	4	fl_size	- размер файла в байтах
(+21)	4	offset	- текущее смещение внутри файла в байтах
(+25)	2	reserv2	- зарезервировано
(+27)	2	last_clu	- номер только что прочитанного кластера
(+29)	3	reserv3	- зарезервировано
(+32)	11	filename	- имя файла в формате FCB (имя выравнено на левую границу поля, дополнено пробелами до 8 символов, справа к нему прилежит 3 символа расширения без точки)
(+43)	2	reserv4	- зарезервировано
(+45)	2	ownr_psp	- PSP программы, открывшей файл
(+47)	2	reserv5	- зарезервировано

Формат таблицы файлов для DOS 4.x:

(0)	4	next	- указатель на следующую таблицу файлов
(+4)	2	file_count	- количество файлов в этой таблице
---- Дальше идут блоки DFCB в количестве file_count штук ----			
(0)	2	handl_num	- количество файловых чисел, связанных с данным файлом (file handle)
(+2)	1	access_mode	- режим доступа к файлу, заданный при открытии файла
(+3)	2	reserv1	- зарезервировано
(+5)	2	dev_info	- информация IOCTL, полученная для устройства, на котором расположен этот файл (подробно формат и назначение этого поля будут рассмотрены в главе, посвященной драйверам)
(+7)	4	driver	- указатель на драйвер, обслуживающий устройство, содержащее файл
(+11)	2	first_clu	- номер первого кластера, распределенного файлу
(+13)	2	time	- время последнего изменения файла в упакованном формате
(+15)	2	date	- дата последнего изменения файла в упакованном формате
(+17)	4	fl_size	- размер файла в байтах
(+21)	4	offset	- текущее смещение внутри файла в байтах
(+25)	2	reserv2	- зарезервировано
(+27)	2	reserv7	- зарезервировано
(+29)	3	reserv3	- зарезервировано
(+32)	1	reserv4	- зарезервировано
(+33)	11	filename	- имя файла в формате FCB (имя выравнено на левую границу поля, дополнено пробелами до 8 символов, справа к нему прилежит 3 символа расширения без точки)
(+44)	2	reserv5	- зарезервировано
(+46)	2	ownr_psp	- PSP программы, открывшей файл
(+48)	2	reserv6	- зарезервировано
(+50)	2	last_clu	- номер только что прочитанного кластера
(+52)	4	reserv8	- зарезервировано

Заголовок драйвера:

(0)	4	next	- указатель на заголовок следующего драйвера. Если смещение адреса следующего драйвера равно FFFF, это последний драйвер в цепочке
(+4)	2	attrib	- атрибуты драйвера
(+6)	2	strateg	- смещение программы стратегии драйвера
(+8)	2	interrupt	- смещение программы обработки прерывания для драйвера
(+10)	8	dev_name	- имя устройства для символьных устройств или количество обслуживаемых устройств для блочных устройств

Формат элемента массива дисковой информации:

(0)	64	path	- текущий путь доступа для диска
(+64)	2	reserv1	- зарезервировано
(+66)	2	reserv2	- зарезервировано
(+68)	1	reserv3	- зарезервировано
(+69)	4	ddcb	- адрес соответствующего DDCB
(+73)	2	cdir_clu	- первый кластер текущего каталога на диске. 0 соответствует корневому каталогу, -1 - если к диску еще не обращались
(+75)	2	reserv4	- зарезервировано
(+77)	2	reserv5	- зарезервировано
(+79)	2	reserv6	- зарезервировано
----- DOS 4.x -----			
(+81)	7	reserv7	- зарезервировано

Заголовок EXE-файла:

(0)	2	signature	- два байта 'MZ' (4Dh, 5Ah), идентифицирующие файл в формате EXE
(+2)	2	part_pag	- длина последней страницы программы в байтах (страница содержит 512 байт)
(+4)	2	file_size	- размер программы в страницах по 512 байт
(+6)	2	rel_item	- число элементов в таблице расположения сегментов
(+8)	2	hdr_size	- размер заголовка файла в параграфах (длина параграфа - 16 байт)

(+10)	2	min_mem	- минимальное количество памяти в параграфах, которое нужно зарезервировать в памяти за концом загруженной программы
(+12)	2	max_mem	- максимальное количество памяти в параграфах, которое нужно зарезервировать в памяти за концом загруженной программы
(+14)	2	ss_reg	- величина смещения от начала программы, которая используется для загрузки сегментного регистра стека SS
(+16)	2	sp_reg	- величина смещения от начала программы, которая используется для загрузки регистра SP
(+18)	2	chk_summ	- контрольная сумма всех слов в файле
(+20)	2	ip_reg	- значение для регистра IP, которое будет использовано при начальном запуске программы
(+22)	2	cs_reg	- смещение от начала программы для установки сегментного регистра кода CS
(+24)	2	reft_off	- смещение от начала файла таблицы расположения сегментов программы
(+26)	2	overlay	- номер оверлея, равен 0 для основного модуля

*
* **Акционерное общество** *
* **"ДИАЛОГ-МИФИ"** *
*
* **ПОСТАВЛЯЕТ ВЫЧИСЛИТЕЛЬНУЮ** *
* **ТЕХНИКУ И ОРГТЕХНИКУ ЗА РУБЛИ И СКВ** *
*
* **СПЕЦИФИКАЦИЯ ПО ЖЕЛАНИЮ ЗАКАЗЧИКА** *
*
* СССР, 115409 Москва, ул. Москворечье, 31-2 *
* Телефон: (095) 3243055, (095) 3247166 *
* Телефакс: (095) 3243055 *

ПРИЛОЖЕНИЕ 2

СОДЕРЖИМОЕ ФАЙЛА sysp.h

```
/* SYSP.H - include-файл для примеров, приведенных в книге */
```

```
/**
*.Name      FP_MAKE
*.Title     Макро для составления FAR-указателя
*.Descr     Макро составляет FAR-указатель, пользуясь
*.          значениями сегмента и смещения
*.Params    FP_MAKE(seg,off)
*.          seg - сегмент;
*.          off - смещение
*.Return    FAR-указатель seg:off
*/
```

```
#define FP_MAKE(seg,off) ((void far *) \
    (((unsigned long) (unsigned)(seg)) << 16L) | \
    ((unsigned long) (unsigned) (off))))
```

```
/* Структура векторной таблицы связи DOS */
```

```
#pragma pack(1)
```

```
typedef struct _CVT_ {
    unsigned mcb_seg;
    void far *dev_cb;
    void far *file_tab;
    void far *clock_dr;
    void far *con_dr;
    unsigned max_btbl;
    void far *disk_buf;
    void far *drv_info;
    void far *fcb_tabl;
    unsigned fcb_size;
    unsigned char num_bdev;
    unsigned char lastdriv;
} CVT;
```

```
/* Блок управления памятью MCB */
```

```
typedef struct _MCB_ {
    unsigned char type;
    unsigned owner;
    unsigned size;
    char reserve[11];
} MCB;
```



```
/* Префикс программного сегмента PSP */
```

```
typedef struct _PSP_ {  
    unsigned char int20h[2];  
    unsigned mem_top;  
    unsigned char reserv1;  
    unsigned char call_dsp[5];  
    void far *term_adr;  
    void far *cbrk_adr;  
    void far *crit_err;  
    unsigned parn_psp;  
    unsigned char file_tab[20];  
    unsigned env_seg;  
    void far *ss_sp;  
    unsigned max_open;  
    void far *file_tba;  
    unsigned char reserv2[24];  
    unsigned char disp[3];  
    unsigned char reserv3[9];  
    unsigned char fcb1[16];  
    unsigned char fcb2[20];  
    unsigned char p_size;  
    unsigned char parm[127];  
} PSP;
```

```
/* Блок управления устройством DOS */
```

```
typedef struct _DDCB_ {  
    unsigned char drv_num;  
    unsigned char drv_numd;  
    unsigned sec_size;  
    unsigned char clu_size;  
    unsigned char clu_base;  
    unsigned boot_siz;  
    unsigned char fat_num;  
    unsigned max_dir;  
    unsigned data_sec;  
    unsigned hi_clust;  
    unsigned char fat_size;  
    char reserv1;  
    unsigned root_sec;  
    void far *drv_addr;  
    unsigned char media;  
    unsigned char acc_flag;  
    struct _DDCB_ far *next;  
    unsigned reserv2;  
    unsigned built;  
} DDCB;
```

```
/* Управляющий блок DOS для файлов */
```

```
typedef struct _DFCB_ {  
    unsigned handl_num;  
    unsigned char access_mode;
```



```
    unsigned reserv1;
    unsigned dev_info;
    void far *driver;
    unsigned first_clu;
    unsigned time;
    unsigned date;
    unsigned long fl_size;
    unsigned long offset;
    unsigned reserv2;
    unsigned reserv7;
    unsigned reserv3;
    char reserv4;
    char filename[11];
    char reserv5[6];
    unsigned ownr_psp;
    unsigned reserv6;
    unsigned last_clu;
    char reserv8[4];
} DFCB;

/* Таблица файлов DOS */
typedef struct _DFT_ {
    struct _DFT_ far *next;
    unsigned file_count;
    DFCB dfcb;
} DFT;

/* Управляющий блок дискового буфера BCB */
typedef struct _BCB_ {
    struct _BCB_ far *next;
    unsigned char drive;
    unsigned char flag;
    unsigned sect_num;
    unsigned reserv1;
    DDCB far *ddcb;
    unsigned reserv2;
} BCB;

/* Информация о диске */
typedef struct _DINFO_ {
    char path[64];
    unsigned reserv1;
    unsigned reserv2;
    unsigned char reserv3;
    DDCB far *ddcb;
    unsigned cdir_clu;
    unsigned reserv4;
    unsigned reserv5;
    unsigned reserv6;
    unsigned char reserv7[7];
} DINFO;
```



```
/* Заголовок EXE-программы */
```

```
typedef struct _EXE_HDR_ {  
    unsigned signature;  
    unsigned part_pag;  
    unsigned file_size;  
    unsigned rel_item;  
    unsigned hdr_size;  
    unsigned min_mem;  
    unsigned max_mem;  
    unsigned ss_reg;  
    unsigned sp_reg;  
    unsigned chk_summ;  
    unsigned ip_reg;  
    unsigned cs_reg;  
    unsigned relt_off;  
    unsigned overlay;  
} EXE_HDR;
```

```
/* таблица расположения сегментов EXE-программы */
```

```
typedef struct _RELOC_TAB_ {  
    unsigned offset;  
    unsigned segment;  
} RELOC_TAB;
```

```
#pragma pack()
```

```
void far *get_cvt(void); /* получить адрес векторной таблицы  
связи */
```

```
CVT far *get_mcvf(void); /* получить адрес векторной таблицы  
связи */
```

```
MCB far *get_fmcb(CVT far *); /* получить адрес первого MCB */  
MCB far *get_nmcb(MCB far *); /* получить адрес  
следующего MCB */
```

```
DDCB far *get_fddcb(CVT far *); /* получить адрес  
первого DDCB */
```

```
DDCB far *get_nddcb(DDCB far *); /* получить адрес  
следующего DDCB */
```

```
DDCB far *get_ddcb(unsigned char); /* получить адрес DDCB  
для диска */
```

```
DFT far *get_fdft(CVT far *); /* получить адрес первой DFT */  
DFT far *get_ndft(DFT far *); /* получить адрес  
следующей DFT */
```

```
BCB far *get_fbcf(CVT far *); /* получить адрес первого BCB */  
BCB far *get_nbcf(BCB far *); /* получить адрес  
следующего BCB */
```

```
int get_exeh(EXE_HDR *, RELOC_TAB **, FILE *); /* прочитать  
заголовок EXE */
```


КОНТРОЛЛЕР ПРЕРЫВАНИЙ 8259

Программируемый контроллер прерываний 8259 (отечественный аналог - КР1810ВН59А) предназначен для обработки до восьми приоритетных уровней прерываний. Возможно каскадирование микросхем, при этом общее число уровней прерываний будет достигать 64.

Контроллер 8259 имеет несколько режимов работы, которые устанавливаются программным путем. В персональных компьютерах XT и AT за первоначальную установку режимов работы микросхем 8259 отвечает BIOS. У программиста скорее всего не возникнет потребность перепрограммировать контроллер - это небезопасно, т. к. неправильное программирование контроллера приведет к нарушению логики работы всей системы.

Однако часто возникает необходимость изменения текущего режима работы (запрет или разрешение прерываний определенного или всех уровней, обработка конца прерывания) или опроса состояния внутренних регистров контроллера. Для этого необходимо ознакомиться со справочными данными на микросхему 8259, где детально описано как первоначальное программирование контроллера, так и управление им во время работы.

Каждому приоритетному уровню прерывания микросхема ставит в соответствие определенный, задаваемый программно номер прерывания. В разделе книги, посвященном особенностям обработки аппаратных прерываний, приводится такое соответствие для машин типа XT и AT.

Если контроллеры 8259 каскадированы, то ведомой микросхеме присваивается код (выдачей в микросхему соответствующего командного слова). Этот код равен номеру входа IRQ ведущей микросхемы, с которым соединен выход запроса прерывания INT ведомой микросхемы. Внутри микросхемы приоритет зависит от номера IRQ и задается программно. Для компьютеров XT и AT самым высоким приоритетом внутри группы, обслуживаемой каждым контроллером, является вход IRQ0. Однако возможно программное изменение приоритетов в рамках так называемого приоритетного кольца. При этом дно приоритетного кольца имеет самый низкий приоритет.

Приведем возможные варианты задания приоритетов:

<i>Вход</i>	<i>Уровни приоритета</i>
IRQ0	7 6 5 4 3 2 1 0
IRQ1	0 7 6 5 4 3 2 1
IRQ2	1 0 7 6 5 4 3 2
IRQ3	2 1 0 7 6 5 4 3
IRQ4	3 2 1 0 7 6 5 4
IRQ5	4 3 2 1 0 7 6 5
IRQ6	5 4 3 2 1 0 7 6
IRQ7	6 5 4 3 2 1 0 7

Наиболее высокий приоритет у входа IRQ с обозначением 0 приоритетного кольца, наиболее низкий - с обозначением 7.

Для обработки прерываний контроллер имеет несколько внутренних регистров. Это регистр запросов прерываний IRR, регистр обслуживания прерываний ISR, регистр маски прерываний IMR. В регистре IRR хранятся запросы на обслуживание прерываний от аппаратуры. После выработки сигнала прерывания центральному процессору соответствующий разряд регистра ISR устанавливается в единичное состояние, что блокирует обслуживание всех запросов с равным или более низким приоритетом. Устранить эту блокировку можно либо сбросом соответствующего бита в ISR, либо командой специального маскирования.

Имеется два типа команд, посылаемых программой в контроллер 8259 - команды инициализации и команды операции. Возможны следующие операции:

- индивидуальное маскирование запросов прерывания
- специальное маскирование обслуженных запросов
- установка статуса уровней приоритета (по установке исходного состояния, по обслуженному запросу, по указанию)
- операции конца прерывания (обычный конец прерывания, специальный конец прерывания, автоматический конец прерывания)
- чтение регистров IRR, ISR, IMR

Мы не будем подробно описывать команды инициализации контроллера 8259, т. к. программистам они скорее всего не понадобятся. Желающих разобраться во всех тонкостях задания начального режима работы контроллера прерываний мы отсылаем к справочной литературе по микросхеме 8259 или ее отечественному аналогу.

Рассмотрим команды операций. Существуют три типа команд операций:

1. Маскирование запросов прерывания.
2. Команды обработки конца прерывания.
3. Опрос регистров и специальное маскирование.

Байты команды маскирования запросов прерывания выводятся соответственно в порты 21h и A1h для первого и второго контроллера 8259 компьютера AT. Команды операций второго и третьего типа используют порты с адресами 20h и A0h. Маскирование запросов прерываний мы уже описывали в главе, посвященной прерываниям. Для маскирования какого-либо уровня прерывания надо записать в регистр маски IMR по адресу 21h или A1h единицу в соответствующий разряд регистра.

Команды обработки конца прерывания приведем в виде таблицы:

<i>Биты байта команды</i>								<i>Описание</i>
<i>D7</i>	<i>D6</i>	<i>D5</i>	<i>D4</i>	<i>D3</i>	<i>D2</i>	<i>D1</i>	<i>D0</i>	
0	0	1	0	0	0	0	0	Обычный конец прерывания
0	1	1	0	0	B2	B1	B0	Специальный конец прерывания, B0...B2 - двоично-десятичный код сбрасываемого разряда в регистре обслуживания прерывания ISR
1	0	1	0	0	X	X	X	Циклический сдвиг уровней приоритета с обычным концом прерывания. Дно приоритетного кольца устанавливается по обслуженному запросу
1	1	1	0	0	B2	B1	B0	Циклический сдвиг уровней приоритета со специальным концом прерывания, B0...B2 - двоично-десятичный код дна приоритетного кольца
1	0	0	0	0	X	X	X	Разрешение вращения уровней приоритета
0	0	0	0	0	X	X	X	Сброс разрешения вращения уровней приоритета
1	1	0	0	0	B2	B1	B0	Циклический сдвиг уровней приоритета без завершения прерывания, B0...B2 - двоично-десятичный код дна приоритетного кольца

Команды третьего типа выдаются также в порты с адресами 20h и A0h. Они имеют следующий формат:

<i>Биты байта команды</i>								<i>Описание</i>
<i>D7</i>	<i>D6</i>	<i>D5</i>	<i>D4</i>	<i>D3</i>	<i>D2</i>	<i>D1</i>	<i>D0</i>	
0	0	0	0	1	1	X	X	Установка режима опроса
0	0	0	0	1	0	1	1	Разрешение чтения регистра ISR
0	0	0	0	1	0	1	0	Разрешение чтения регистра IRR
0	1	1	0	1	0	0	0	Разрешение триггера специального маскирования
0	1	0	0	1	0	0	0	Сброс триггера специального маскирования

По команде обычного конца прерывания устанавливается в нулевое состояние разряд ISR, соответствующий последнему обслуженному запросу.

Команда специального конца прерывания уставнавливает в нулевое состояние тот разряд ISR, номер которого указан в разрядах B0...B2 команды.

Команда циклического сдвига уровней приоритета с обычным концом прерывания устанавливает в ноль разряд ISR, соответствующий последнему обслуженному запросу, и этому же номеру запроса присваивается низший уровень приоритета.

Аналогично работает команда циклического сдвига уровней приоритета со специальным концом прерывания, только низший уровень приоритета присваивается тому входу IRQ, номер которого указан в разрядах B0...B2 команды.

Команда циклического сдвига уровней приоритета устанавливает статус уровней приоритета без выполнения операции конца прерывания. Разряды В0...В2 указывают дно приоритетного кольца.

После выполнения команд разрешения чтения регистров ISR или IRR при выполнении команды ввода из порта 20h и A0h считывается соответственно содержимое регистров ISR и IRR. Для получения содержимого регистра IMR необходимо выполнить чтение портов с адресами соответственно 21h и A1h.

Команда разрешения триггера специального маскирования блокирует действие тех разрядов ISR, которые замаскированы командой типа 1 (маскирования индивидуальных приоритетных уровней запроса прерывания). Специальное маскирование используется для обслуживания такого запроса, который блокируется старшим или равным по уровню приоритета обслуженным запросом, хранящимся в ISR, не сбрасывая последний.

Чтение регистров ISR и IRR может использоваться резидентными программами при проверке возможности своей активизации - можно проверить, не выполняется ли в настоящий момент обработка какого-нибудь прерывания, которая может конфликтовать с действиями резидентной программы.

Акционерное общество
"ДИАЛОГ-МИФИ"

•

**ПРИНИМАЕТ ЗАЯВКИ
НА ПУБЛИКАЦИЮ РЕКЛАМЫ В СВОИХ ИЗДАНИЯХ**

СССР, 115409 Москва, ул. Москворечье, 31-2
Телефон: (095) 3243055, (095) 3247166
Телефакс: (095) 3243055

ГЛАВА 1

ИНТЕРФЕЙС МЕЖДУ ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ И АППАРАТУРОЙ

Фирмы-разработчики аппаратного обеспечения постоянно совершенствуют внешние устройства и другие узлы персонального компьютера. Постоянно появляются новая периферийная аппаратура и новые модификации уже существующих устройств. Старые устройства наделяются новыми возможностями, новые делают такое, о чем раньше не приходилось и мечтать.

Этот процесс можно только приветствовать, однако что делать с уже разработанным программным обеспечением, рассчитанным на старую аппаратуру? Если разработка программного обеспечения была дорогостоящая, если по своим возможностям оно еще не устарело, вряд ли целесообразно переделывать все заново только из-за того, что появился новый дисковод с большей емкостью носителя данных или новый принтер, обеспечивающий лучшее качество печати.

Интуитивно ясно, что должна существовать какая-то программная прослойка между аппаратным и программным обеспечением, выполняющая "согласующие" и "унифицирующие" действия. Эта прослойка работает напрямую с аппаратурой, а прикладное (да и системное) программное обеспечение имеет дело только с этой интерфейсной прослойкой.

В разных типах компьютеров и для разных типов операционных систем существуют различные способы устранения зависимости программного обеспечения от аппаратуры. По-видимому, одним из наиболее неудачных следует считать способ, примененный в операционной системе ОС ЕС для ЭВМ ряда ЕС. Для ЭВМ ряда ЕС любое периферийное устройство имеет одинаковый и достаточно сложный аппаратный интерфейс с каналами ЭВМ (канал - это устройство, обеспечивающее ввод/вывод информации в ЭВМ ряда ЕС). Для каждого типа устройства составляются каналные программы (программы, которые будут выполняться каналом). Прикладное программное обеспечение может пользоваться либо вводом/выводом на этом, крайне низком, канальном уровне, либо осуществлять ввод/вывод на уровне так называемых методов доступа. Библиотеки программ методов доступа содержат программы, работающие на канальном уровне. Для любого, даже очень простого, нового устройства в этой системе необходим сложный аппаратный интерфейс с каналом. Составление программ своего метода доступа - крайне сложная задача, доступная лишь профессионалам высокого класса. О внесении изменений в существующие программы методов доступа говорить вообще вряд ли стоит, т. к. они очень сложны, а исходные тексты этих программ, как правило, отсутствуют.

Другой подход используется в операционных системах UNIX и аналогичных ей (XENIX, VENIX и т. д.).

В этих операционных системах для каждого устройства составляется своя программа обслуживания - драйвер. Эта программа выполняет все операции со

своим устройством. Сама операционная система имеет дело только с драйвером, а прикладное программное обеспечение для выполнения ввода/вывода вызывает соответствующие модули операционной системы. Такая программная буферизация устраняет, с одной стороны, зависимость операционной системы от аппаратуры, с другой - зависимость прикладной программы от операционной системы и от аппаратуры.

Для каждого нового устройства (например, для нового видеоконтроллера) фирма-разработчик предоставляет свой драйвер. Если вы купили новое устройство, вы просто подключаете новый драйвер к операционной системе, и все ваши любимые старые программы будут теперь работать с новым устройством.

Этот подход имеет тот недостаток, что вы не сможете легко и свободно переносить вашу операционную систему с одного типа компьютера на другой, даже совместимый с ним. Драйверы операционной системы UNIX очень сильно привязаны к аппаратуре. На компьютере, который совместим с исходным не полностью, эти драйверы работать не будут.

Обычно операционные системы поставляются с конкретным комплектом драйверов под определенный тип компьютера, и с этим приходится считаться.

Операционная система MS-DOS, работающая на компьютерах фирмы IBM или совместимых с ними, тоже использует механизм драйверов.

Однако драйверы MS-DOS не всегда обращаются напрямую к аппаратуре. Обычно они вызывают функции BIOS, и уже BIOS выполняет все действия по вводу/выводу. Конечно, BIOS содержит программы обслуживания только стандартных устройств ввода/вывода, нестандартные устройства обслуживаются драйверами напрямую.

Использование BIOS как дополнительного интерфейса между драйверами стандартных устройств и аппаратурой резко повышает "живучесть" MS-DOS на не вполне совместимых с IBM персональных компьютерах. И это действительно так - самая распространенная на сегодняшний день операционная система MS-DOS версии 3.30 работает на всех компьютерах, хоть сколько-нибудь совместимых с IBM PC.

Это возможно благодаря тому, что производители совместимых компьютеров учитывают в программах BIOS все аппаратные особенности, и DOS "не видит" отличий. А прикладная программа - тем более.

Почему же этот способ не используется в операционных системах UNIX или OS/2? Дело в том, что, к сожалению, программы BIOS не являются реентерабельными. Это не имеет значения для однозадачной MS-DOS, а мультизадачные операционные системы вынуждены сами организовывать обслуживание аппаратуры реентерабельным способом. (Существуют еще проблемы разделения ресурсов между параллельно выполняющимися процессами, которые тоже не решаются в рамках BIOS.)

Таким образом, независимость аппаратного и программного обеспечения в DOS обеспечивается, с одной стороны, BIOS для стандартных устройств, с другой стороны - драйверами.

Пользователи могут легко дополнять операционную систему своими драйверами, составленными для нестандартных устройств. Возможна также замена стандартных драйверов, замена или расширение функций BIOS.

В разделе, посвященном резидентным программам, мы уже рассказывали о том, как можно организовать обслуживание нестандартных устройств в DOS с помощью TSR-программ. Но это "незаконный" способ с точки зрения DOS, т. к. при этом DOS не сможет использовать свои стандартные средства ввода/вывода, ориентированные на связь с аппаратурой через драйверы. Поэтому наилучшим способом организации обслуживания аппаратуры является использование драйвера.

ГЛАВА 2

СТРУКТУРА ЗАГРУЖАЕМОГО ДРАЙВЕРА

Драйвер - это еще одна разновидность программ в дополнение к уже изученным нами программам формата COM и EXE. Иногда говорят, что драйверы - это разновидность COM-программ, но это не так. Скорее, способ получения загрузочного модуля драйвера похож на способ получения программы в формате COM. Есть еще одно сходство драйверов и программ в формате COM (которое как раз и появляется из-за одинакового способа их получения) - загрузочные модули этих программ являются точным отображением исходного текста на языке ассемблера без добавления каких-либо управляющих блоков в начало файла, как это происходит в программах формата EXE. (К сожалению, драйвер должен быть написан на языке ассемблера. Авторам этой книги не известны способы составления драйверов на других языках программирования.)

Но оказывается, управляющий блок в самом начале модуля драйвера имеется. Это так называемый заголовок драйвера. Но в отличие от программ формата EXE этот заголовок создается не редактором связи, а самим программистом и должен быть помещен в самое начало исходного текста программы-драйвера.

При загрузке драйвера в память заголовок драйвера тоже помещается в оперативную память и в нем операционная система производит некоторые изменения, о которых мы еще будем говорить.

Таким образом, можно говорить и о сходстве драйвера с программами в формате EXE, т. к. в начале загрузочного модуля драйвера имеется управляющий блок, только этот управляющий блок в отличие от заголовка EXE-файла является принадлежностью самой программы и загружается вместе с ней в память. Заголовок EXE-программы используется при загрузке EXE-программы, но после загрузки операционная система убирает его из памяти.

Не стоит пытаться запускать драйвер как программу в формате COM, т. к. управление будет передано в область памяти, содержащую заголовок драйвера,

а там нет правильных машинных команд. Поэтому обычно файлы драйверов имеют расширения имени, отличные от COM или EXE. Чаще всего используются расширения SYS, DRV, иногда BIN. На самом деле расширение имени можно задавать любое, т. к. при описании драйвера в файле CONFIG.SYS указывается его полное имя.

Для драйвера никогда не создается префикс программного сегмента PSP. В начале исходного текста программы-драйвера не ставится директива ORG 100H (в COM-программе она ставится), т. к. не надо резервировать место для PSP.

Что же представляет из себя загрузочный модуль драйвера?

Как уже было сказано, в начале модуля находится заголовок драйвера. Мы уже немного говорили о нем при описании векторной таблицы связи операционной системы. Приведем формат заголовка:

(0)	4	next	- указатель на заголовок следующего драйвера. Если смещение адреса следующего драйвера равно FFFF, это последний драйвер в цепочке
(+4)	2	attrib	- атрибуты драйвера
(+6)	2	strateg	- смещение программы стратегии драйвера
(+8)	2	interrupt	- смещение программы обработки прерывания для драйвера
(+10)	8	dev_name	- имя устройства для символьных устройств или количество обслуживаемых устройств для блочных устройств

Как уже было сказано, все драйверы связаны в цепочку. Самый первый драйвер находится сразу за векторной таблицей связи. Поле next заголовка драйвера указывает на следующий драйвер (на его заголовок). Это поле имеет формат DWORD-указателя и состоит из компоненты адреса сегмента и смещения. Признаком того, что данный драйвер последний в цепочке, служит значение 0FFFFh в компоненте смещения поля next.

Программист, когда он составляет программу-драйвер, заносит в это поле либо 0FFFFh:0FFFFh, если исходный текст содержит только один драйвер, либо адрес следующего драйвера (в виде дальней ссылки на метку заголовка следующего драйвера). Если исходный текст содержит несколько драйверов, то в заголовке последнего в поле next должно находиться значение 0FFFFh:0FFFFh.

При загрузке драйверов в память операционная система изменит содержимое поля next в заголовках драйверов для того, чтобы это поле указывало на заголовок следующего драйвера в цепочке. Изменит в памяти, а не в файле драйвера!

Обычно исходный текст программы содержит один драйвер и поле next задается следующим образом:

```
next DD 0FFFFFFFFh
```

Следующее поле в заголовке драйвера - поле атрибутов драйвера attrib.

Это поле описывает устройство, обслуживаемое данным драйвером. Каждый бит слова отвечает за ту или иную особенность устройства. Перед детальным рассмотрением назначения каждого бита этого слова заметим, что бит 15 (самый старший бит) указывает, является ли это устройство символьным или блочным.

Следует специально отметить, что все драйверы можно разделить на две группы - драйверы символьных устройств и драйверы блочных устройств. Первые обслуживают устройства посимвольного ввода/вывода, такие, как принтеры, клавиатура, контроллеры последовательной передачи данных RS232S и т. д., вторые ориентированы на ввод/вывод блоками - это диски.

Как правило, все нестандартные устройства, начиная от цифрового вольтметра до роботов и систем автоматизации производственных процессов, обслуживаются символьными драйверами. Хотя этот тип драйверов ориентирован на передачу данных посимвольно, для быстродействующих устройств ввода/вывода можно организовать буферизацию (средствами операционной системы).

Блочные драйверы могут вам потребоваться в основном для обслуживания своих нестандартных дисковых устройств. Например, можно использовать более плотную запись информации на дискетах для повышения их емкости, можно через аппаратуру связи персонального компьютера и ЭВМ серии ЕС создавать псевдо-винчестеры на дисках ЕС (такие "винчестеры" будут восприниматься DOS как обычные стандартные диски). С помощью блочных драйверов можно организовать защиту информации на дисках от несанкционированного доступа, если ваш драйвер будет шифровать записываемую на диск информацию и предоставлять ее по предъявлению пароля.

Мы рассмотрим оба типа драйверов, каждый раз будем отмечать особенности символьных и блочных устройств.

Приведем назначение битов слова атрибутов символьного драйвера:

Бит	Назначение
0	1 - драйвер обслуживает стандартное устройство ввода; 0 - этот драйвер не обслуживает стандартное устройство ввода
1	1 - драйвер обслуживает стандартное устройство вывода; 0 - драйвер не обслуживает стандартное устройство вывода
2	1 - это драйвер стандартного устройства NUL; 0 - драйвер не обслуживает устройство NUL
3	1 - драйвер обслуживает часы
4	Зарезервировано, бит должен быть равен 0
5	Зарезервировано, бит должен быть равен 0
6	1 - разрешено использование драйвером функций GENERIC IOCTL (для версий DOS, более поздних, чем 3.2); 0 - функции GENERIC IOCTL не поддерживаются
7-10	Эти биты зарезервированы и должны быть равны 0

11	1 - для символьных устройств поддерживаются функции открытия/закрытия устройства (OPEN/CLOSE); 0 - для символьных устройств функции OPEN/CLOSE не поддерживаются
12	Зарезервировано, бит должен быть равен 0
13	1 - для символьных устройств поддерживается функция вывода до получения состояния занятости устройства; 0 - функция вывода до состояния занятости не поддерживается
14	1 - поддерживаются функции IOCTL 0 - функции IOCTL не поддерживаются
15	1 - символьное устройство 0 - блочное устройство

Для драйверов блочных устройств формат слова атрибутов другой:

Бит	Назначение
0	Зарезервировано, бит должен быть равен 0
1	1 - драйвер поддерживает 32-битовую адресацию сектора (для версий DOS, начиная с 4.00 и более поздних); если установлен этот бит, поле номера сектора всех запросов является двойным словом, добавляемым в конец заголовка запроса, старое поле номера сектора должно содержать -1); 0 - используется 16-битовая адресация сектора
2-5	Эти биты зарезервированы и должны быть равны 0
6	1 - поддерживаются логические устройства (используются блочными драйверами для управления "виртуальными" флоппи-дисками, создаваемыми драйвером DRIVER.SYS в MS-DOS версии 3.2 и более поздних версиях); 0 - логические устройства для блочных драйверов не поддерживаются
7-10	Эти биты зарезервированы и должны быть равны 0
11	1 - единица в этом бите означает, что драйвер поддерживает функцию проверки замены носителя данных в устройстве (например, замены дискеты): используется для MS-DOS версий 3.00 и более поздних; 0 - для блочных устройств функция проверки замены носителя данных не поддерживается
12	Зарезервировано, бит должен быть равен 0
13	1 - драйвер не использует стандартное IBM-устройство, и необходимо выдать запрос на построение блока параметров BIOS BPB; 0 - используется IBM-устройство
14	1 - поддерживаются функции IOCTL; 0 - функции IOCTL не поддерживаются
15	1 - символьное устройство; 0 - блочное устройство

Более подробно назначение отдельных битов слова атрибутов драйвера мы рассмотрим позже, при описании функций, выполняемых драйвером. Приведем пример определения слова атрибутов для символьного драйвера нестандартного устройства с минимальными функциональными возможностями:

```
attrib DW      8000h
```

После слова атрибутов драйвера находятся два очень важных поля: смещение программы стратегии драйвера `strateg` и смещение программы обработки прерывания `interrupt`.

Эти две программы используются DOS для организации обращения к драйверу. Для обращения к драйверу DOS формирует в своей области данных запрос, состоящий из заголовка стандартного формата и переменной части запроса, длина и формат которой зависят от типа запроса. После этого DOS считывает из заголовка драйвера значение смещения программы стратегии и передает ей управление, записав в регистры `ES:BX` адрес заголовка запроса.

Задача программы стратегии - запомнить этот адрес внутри тела драйвера для дальнейшего использования или организовать очередь запросов обслуживания.

Сразу после вызова программы стратегии DOS вызывает программу обработки прерываний, определив ее адрес из поля `interrupt` заголовка драйвера.

Программа обработки прерывания извлекает только что записанный программой стратегии адрес заголовка запроса и выполняет ту функцию, номер которой записан в запросе. Номер функции находится в заголовке запроса.

Результаты выполнения функции программа прерывания записывает в специально отведенные поля заголовка запроса, и на этом процедура обращения DOS к драйверу завершается.

Формат заголовка запроса будет приведен ниже, а сейчас покажем, как в заголовке драйвера задаются смещения программ стратегии и прерывания:

```
strateg  DW      strateg_proc
interrupt DW      interrupt_proc
```

Последнее поле заголовка драйвера `dev_name` имеет различную интерпретацию для символьных и блочных устройств.

Для символьных устройств в этом поле должно располагаться выравненное по левому краю и дополненное пробелами до восьми символов имя устройства. Это имя будет использоваться для обращения к драйверу. Если вы собираетесь заменить драйвер стандартного символьного устройства DOS на свой, вы должны записать имя устройства заглавными буквами:

```
dev_name  DB      'AUX      '
```

Для блочных устройств первый байт поля `dev_name` содержит количество устройств, обслуживаемых данным драйвером, остальные семь байт не используются:

```
dev_name  DB      1
          DB      7 dup(?)
```


Таким образом, мы выяснили, что драйвер содержит в самом начале заголовок, и где-то дальше должны располагаться программы стратегии и прерывания. (Не следует путать программу прерывания драйвера с программой обслуживания аппаратных или программных прерываний. Хотя программа прерывания драйвера немного похожа на обработчик программных прерываний, назначение этой программы и механизм ее использования совершенно другие.)

Что еще может находиться в программе-драйвере?

Это могут быть области данных, используемые драйвером, и подпрограммы, вызываемые программами стратегии и прерывания. Иногда стандартные драйверы переназначают на себя некоторые векторы прерываний, и тогда они содержат в себе обработчики этих прерываний. В области памяти, отведенной операционной системой драйверу, может располагаться стек драйвера, если размер системного стека недостаточен.

На длину драйвера накладывается такое же ограничение, как и на длину COM-программ - 64 Кбайта, то есть один сегмент.

ГЛАВА 3

ПРОЦЕСС ЗАГРУЗКИ ДРАЙВЕРОВ

Системный файл DOS IO.SYS содержит некоторые драйверы устройств, составляющие базовую систему ввода/вывода. Эти драйверы появляются в памяти при загрузке операционной системы и связаны в цепочку через поля `next` в своих заголовках. Такие драйверы являются резидентными драйверами операционной системы.

Резидентные драйверы могут быть заменены драйверами пользователя, кроме того, пользователь может добавить в список драйверов новые.

Для подключения драйвера пользователя к операционной системе файл CONFIG.SYS должен содержать команду `DEVICE=<путь файла драйвера> <параметры>`. Например:

```
DEVICE=c:\dos\smartdrv.sys 120
```

В этом примере подключается драйвер `smartdrv.sys`, который находится в каталоге `dos` на диске `c:`. В качестве параметра инициализации драйверу передается число 120. (Параметры считываются драйвером один раз в процессе инициализации драйвера. Об этом мы будем говорить подробно в разделе, посвященном инициализации драйвера.)

В списке драйверов драйверы пользователя находятся перед резидентными. В этом можно убедиться, посмотрев на результаты работы программы `DRI.COM`, описанной ранее. Эта программа выдает весь список драйверов с самого его начала. Для каждого драйвера выводится адрес драйвера в памяти, слово атрибутов драйвера и имя устройства (либо количество обслуживаемых блочным драйвером устройств).

Device Drivers Information V1.00
 Copyright (C)Frolov A.,1990

Address	Attr	Device Name
02C1:0048	8004	NUL
112F:0000	8800	RBUSDRIV
10E4:0000	0800	-----> Block Device, Number of Units: 0001
0D86:0000	C800	SMARTAAR
0CC7:0000	A000	XMSXXXX0
0BA5:0000	6842	-----> Block Device, Number of Units: 0003
0070:016E	8013	CON
0070:0180	8000	AUX
0070:0192	A040	PRN
0070:01A4	8008	CLOCK\$
0070:01B6	0842	-----> Block Device, Number of Units: 0003
0070:01CA	8000	COM1
0070:01DC	A040	LPT1
0070:01EE	A040	LPT2
0070:0200	A040	LPT3
0070:0212	8000	COM2
0070:0224	8000	COM3

Первым в этом списке всегда идет драйвер с именем устройства NUL. Это нуль-устройство, используемое для тестовых целей. Драйвер псевдоустройства NUL не может быть переназначен, он всегда расположен непосредственно за векторной таблицей связи DOS. Далее идут драйверы, описанные в файле CONFIG.SYS следующим образом:

```
device=ssstor.sys
device=e:\C600\BIN\himem.sys
device=e:\c600\bin\smartdrv.sys 530
device=e:\c600\bin\ramdrive.sys 732 512 64 /e
device=e:\vega\rbusdrv.sys 1 80 378 379 37a 37a
```

Начиная с драйвера консоли CON идут резидентные драйверы, имеющие сегментный адрес 0070. Это драйвер последовательного канала связи AUX, драйвер устройства печати PRN, драйвер часов CLOCK\$, драйверы последовательных каналов COM1, COM2, COM3 и драйверы устройств печати LPT1, LPT2, LPT3.

Если ваш драйвер должен заменить стандартный, в поле имени заголовка драйвера укажите имя устройства заглавными буквами (например, LPT1). Система разместит ваш драйвер в цепочке драйверов до стандартного с именем LPT1.

Когда операционная система загружает драйвер, она создает заголовок запроса и помещает в него команду инициализации. Затем вызывается программа стратегии драйвера, которой передается адрес подготовленного заголовка запроса. После этого вызывается программа прерывания драйвера. Эта программа просматривает заголовок запроса, определяет, что пришла команда инициализации, и начинает ее обрабатывать.

Инициализирующая часть выполняется только один раз при загрузке драйвера. Могут выполняться такие действия, как считывание параметров инициализации из файла CONFIG.SYS, выдача инициализирующих команд на обслужи

ваемое устройство ввода/вывода, инициализация внутренних областей данных и т. д. Перед завершением своей работы инициализирующая часть драйвера записывает в заголовок запроса размер резидентной части драйвера. Сама инициализирующая часть больше не будет нужна, поэтому эта часть драйвера не оставляется при инициализации резидентной в памяти.

Таким образом, количество памяти, отводимое драйверу при загрузке, операционная система определяет исходя из той информации, которую сам драйвер передает операционной системе при инициализации, а не пользуется длиной файла драйвера, как это можно было бы предположить.

Более подробно процесс инициализации будет рассмотрен при описании команды инициализации драйвера.

ГЛАВА 4

СВЯЗЬ ДРАЙВЕРА С ОПЕРАЦИОННОЙ СИСТЕМОЙ

Рассмотрим теперь более подробно механизм взаимодействия драйвера и операционной системы.

После загрузки драйвер становится как бы частью операционной системы. Все обращения к драйверу DOS выполняет с использованием заголовка драйвера. Для примера приведем вид заголовка символьного драйвера, выполняющего только простейшие функции:

```
next      DD      0FFFFFFFFh
attrib    DW      8000h
strateg    DW      strateg_proc
interrupt DW      interrupt_proc
dev_name   DB      'TESTDRV'
```

Это символьный драйвер (старший бит поля attrib равен 1), исходный текст содержит только один драйвер (поле next содержит значение 0FFFFFFFFh), имя устройства, которое нужно будет использовать при обращении к драйверу - TESTDRV. Имя устройства не должно совпадать с именем файла, содержащего символьный драйвер, иначе вы не сможете обратиться к файлу, например, для его переименования - DOS будет работать не с файлом, а с устройством.

Как уже было сказано, перед обращением к драйверу DOS подготавливает заголовки запроса в своей области данных и вызывает программу стратегии, извлекая ее смещение из заголовка драйвера. Программа стратегии обычно очень проста, т. к. ее задача - запомнить адрес заголовка запроса в области памяти драйвера. Область для хранения адреса заголовка запроса может быть определена следующим образом:

```
req_off   DW      ?
req_seg    DW      ?
```


Тогда программа стратегии должна записать содержимое регистра ES в поле req_seg, а регистра BX - в поле req_off:

```
strateg_proc:  mov cs:req_off,bx
               mov cs:req_seg,es
               ret
```

Драйвер состоит из одного сегмента кодов, поэтому для адресации данных используется сегментный регистр CS. Запрос операционной системы к драйверу состоит из заголовка, имеющего фиксированный формат и длину 13 байт, и переменной части, размер и формат которой зависит от выполняемой функции.

Приведем формат заголовка запроса:

- | | | | |
|------|---|----------|--|
| (0) | 1 | size | - длина запроса в байтах (длина заголовка запроса плюс длина переменной части запроса) |
| (+1) | 1 | unit | - номер устройства (используется для блочных устройств, указывает, с каким именно устройством, обслуживаемым драйвером, будет работать операционная система) |
| (+2) | 1 | cmd | - код команды, которую требуется выполнить (может иметь значение от 0 до 18h) |
| (+3) | 2 | status | - слово состояния устройства, заполняется драйвером перед возвратом управления операционной системе |
| (+5) | 8 | reserved | - зарезервировано |

После вызова программы стратегии DOS передает управление программе прерывания (без параметров). Задача программы прерывания - выполнить команду, код которой находится в поле cmd заголовка запроса. Если драйвер блочного устройства обслуживает несколько логических устройств, то в поле unit находится номер устройства, для которого необходимо выполнить команду.

В зависимости от выполняемой команды запрос может содержать другую информацию, необходимую для выполнения команды.

Как результаты выполнения команды возвращаются DOS? Данные (или адреса данных), полученные драйвером от физического устройства ввода/вывода, помещаются в область переменной части запроса. Драйвер также должен установить слово состояния устройства status в заголовке запроса в соответствии с результатами выполнения команды. Приведем формат слова состояния устройства:

Бит	Назначение
0-7	Код ошибки устройства (если команда выполнялась с ошибкой и драйвер установил признак ошибки (бит 15) в единицу, в это поле он должен записать код ошибки)
8	Команда выполнена. Этот бит всегда устанавливается драйвером перед тем, как он возвращает управление операционной системе

- 9 Занято. Этот бит устанавливается обработчиком команды, когда физическое устройство занято выполнением предыдущей операции и поэтому не может выполнить требуемую команду. Этот бит используется также для передачи такой информации, как "буфер клавиатуры не пуст", "среда носителя данных заменяемая" (в команде проверки возможности замены среды носителя данных)
- 10-14 Зарезервировано
- 15 Признак ошибки. Устанавливается драйвером, когда он не может обработать запрос или произошла физическая либо логическая ошибка при обработке правильного запроса. Биты 0-7 при этом должны содержать код ошибки

Приведем таблицу возможных кодов ошибок:

Код	Описание
0	Нарушение защиты от записи. Была предпринята попытка записи информации на защищенное от записи устройство
1	Неизвестное устройство
2	Устройство не готово
3	Неизвестная команда. Затребованная команда не поддерживается драйвером
4	Ошибка CRC. При выполнении команды обнаружена ошибка циклического кода проверки
5	Неправильная длина запроса. Поле длины в заголовке запроса содержит неверное значение
6	Ошибка при поиске дорожки (дорожка не найдена)
7	Неизвестный носитель данных
8	Сектор не найден
9	Нет бумаги в принтере
0Ah	Ошибка записи
0Bh	Ошибка чтения
0Ch	Общая ошибка
0Dh	Зарезервировано
0Eh	Зарезервировано
0Fh	Неразрешенная замена диска (только для MS-DOS версии 3.0 и более поздних версий)

Общая схема действий программы прерывания драйвера такова:

- получив управление от операционной системы, программа прерывания сохраняет содержимое всех регистров процессора и считывает номер команды из заголовка запроса

- при необходимости программа считывает дополнительную информацию из области запроса
- затребованная команда выполняется, если она поддерживается драйвером
- если драйвер считывает какие-либо данные от обслуживаемого физического устройства для передачи их DOS, то сами данные или их адреса программа прерывания записывает в область запроса
- программа прерывания устанавливает слово состояния устройства в соответствии с результатами выполнения команды; если драйвер не поддерживает затребованную команду, в слове состояния устанавливаются биты 15 и в биты 0-7 записывается код ошибки 3 "Неизвестная команда"
- восстанавливается содержимое регистров процессора, и управление возвращается операционной системе с помощью команды возврата из дальней процедуры

Приведем фрагмент исходного текста программы прерывания, который выполняет описанные выше функции:

`interrupt_proc:`

```

    push es      ;сохраняем регистры
    push ds
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push bp

```

; Устанавливаем ES:BX на заголовок запроса

```

    mov ax,cs:req_seg
    mov es,ax
    mov bx,cs:req_off

```

; Загружаем в регистр AL код команды из заголовка
; запроса и умножаем его на 2 для получения индекса
; в таблице адресов команд

```

    mov al,es:[bx]+2
    shl al,1

```

```

    sub ah,ah      ; обнуляем AH
    lea di,functions ; DI содержит смещение таблицы команд
    add di,ax      ; добавляем смещение
    jmp word ptr [di] ; переходим по адресу, взятому
                    ; из таблицы

```

`functions LABEL WORD ; это таблица функций`

```

    dw initialize
    dw check_media
    dw make_bpb

```



```

dw    ioctl_in
dw    input_data
dw    nondestruct_in
dw    input_status
dw    clear_input
dw    output_data
dw    output_verify
dw    output_status
dw    clear_output
dw    ioctl_out
dw    Device_open
dw    Device_close
dw    Removable_media

;---выход из драйвера, если функция не поддерживается

check_media:
make_bpb:
ioctl_in:
nondestruct_in:
input_status:
clear_input:
output_verify:
output_status:
clear_output:
ioctl_out:
Removable_media:

; Если функция не поддерживается драйвером, устанавливаем
; в единицу биты 15 (ошибка), 8 (выполнение команды
; завершено). В биты 0-7 записываем код ошибки 3 -
; неизвестная команда.

    or    es:word ptr [bx]+3,8103h
    jmp   quit

;=====

; Это пример обработчика команды:

Device_open:

; . . . . .
; Некоторые действия для открытия устройства.
; . . . . .

    jmp   quit

;=====

;---выходим, модифицируя байт состояния status в заголовке
; запроса

quit:

    or    es:word ptr [bx]+3,100h    ;устанавливаем бит 8
                                     ;(выполнение команды
                                     ;завершено)

```



```

pop bp      ;восстанавливаем регистры
pop di
pop si
pop dx
pop cx
pop bx
pop ax
pop ds
pop es
ret

```

В следующем разделе мы подробно рассмотрим все команды, коды которых могут передаваться драйверу через заголовок запроса. Для каждой команды будет приведен формат области запроса.

ГЛАВА 5

ФУНКЦИИ ЗАГРУЖАЕМОГО ДРАЙВЕРА

Как уже было сказано, номер функции, которую должен выполнить драйвер, передается операционной системой через поле `cmd` заголовка запроса. Рассмотрим отдельные функции.

0 - Инициализация драйвера

Эта функция выполняется только один раз при загрузке драйвера и подключении его к операционной системе.

Функция инициализации должна поддерживаться любым драйвером, т. к. она сообщает операционной системе сведения, необходимые DOS для правильного подключения и использования драйвера.

Приведем формат запроса для команды инициализации:

- | | | | |
|-------|----|----------|---|
| (0) | 13 | header | - заголовок запроса |
| (+13) | 1 | n_units | - количество устройств, обслуживаемых драйвером. Это поле заполняется только блочным драйвером |
| (+14) | 4 | end_addr | - конечный FAR-адрес резидентной части кода драйвера. В это поле драйвер записывает адрес байта памяти, следующего за той частью кода драйвера, которая должна стать резидентной |
| (+18) | 4 | parm | - FAR-адрес строки параметров инициализации драйвера из файла CONFIG.SYS. Эта строка содержит все, что находится в строке файла после команды 'DEVICE=', она заканчивается символами перевода строки и возврата каретки 0Ah, 0Dh
При возврате драйвер блочного устройства должен записать в это поле адрес массива указателей на |

- блоки параметров BIOS (BPB), по одному указателю на каждое устройство, обслуживаемое драйвером
- (+22) 1 drive - номер устройства. Для версии DOS 3.0 и более поздних версий в это поле при загрузке драйвера операционная система заносит номер, назначенный устройству, обслуживаемому драйвером. Например, для устройства A это 0, для B - 1 и т. д.

При инициализации драйвер символьного устройства сохраняет в своей внутренней области данных параметры инициализации, используя адрес parm. Если параметры содержат числовые величины, программа инициализации может произвести их перекодировку и сохранить значения в двоичном формате.

Затем драйвер может выполнить инициализацию обслуживаемого физического устройства ввода/вывода, инициализацию своих внутренних переменных, вывести на экран какие-либо сообщения либо даже запросить у оператора дополнительные данные - функция инициализации может пользоваться для организации диалога с оператором и других действий функциями прерывания 21h с номерами от 01h до 0Ch, 25h, 30h, 35h и функциями BIOS.

Кроме этого, драйвер должен заполнить поле end_addr адресом конца резидентной части драйвера. Т. к. программа инициализации выполняется только один раз, обычно ее располагают в конце драйвера и для экономии памяти не оставляют резидентной.

Драйверы блочных устройств дополнительно должны вернуть DOS количество обслуживаемых устройств (в поле n_units) и указатель на массив указателей на блоки BPB (в поле parm).

Количество устройств используется DOS для определения логических имен устройств. Например, если ваш драйвер обслуживает три логических устройства и на момент его загрузки в системе имеются устройства A, B и C, то устройства, обслуживаемые вашим драйвером, получают имена D, E и F. Количество устройств необходимо указывать также и в заголовке драйвера, в первом байте поля имени устройства dev_name.

Для каждого логического устройства драйвер должен содержать так называемый блок параметров BIOS (BIOS Parameter Block) BPB.

Блок BPB содержится в загрузочном секторе диска и содержит информацию, необходимую BIOS для работы с диском. Приведем формат BPB:

- (0) 2 sect_siz - количество байт в одном секторе диска
- (+2) 1 clustsiz - количество секторов в одном кластере
- (+3) 2 res_sect - количество зарезервированных секторов
- (+5) 1 fat_cnt - количество таблиц FAT
- (+6) 2 root_siz - максимальное количество дескрипторов файлов, содержащихся в корневом каталоге диска

- | | | | |
|-------|---|----------|--|
| (+8) | 2 | tot_sect | - общее количество секторов на носителе данных (в разделе DOS) |
| (+10) | 1 | media | - байт-описатель среды носителя данных |
| (+11) | 2 | fat_size | - количество секторов, занимаемых копией FAT |

Подробно формат и назначение полей BPB будут описаны в разделе, посвященном файловой системе DOS.

Приведем фрагмент исходного текста драйвера, возвращающего при инициализации указатель на массив BPB:

```
lea  dx,bpb_ptr
mov  es:[bx+18],dx
mov  es:[bx+20],cs
```

В этом примере предполагается, что ES:BX содержит адрес заголовка запроса.

Область данных может содержать, например, такое описание bpb_ptr:

```
bpb:      DW    512    ; количество байт на сектор
          DB     1     ; количество секторов на кластер
          DW     1     ; зарезервировано секторов
          DB     2     ; количество копий FAT
          DW    64     ; максимальное количество файлов
                  ; в корневом каталоге
          DW   360     ; общее число секторов на диске
          DB   0FCh    ; описатель среды
          DW     2     ; количество секторов в FAT
;
bpb_ptr   DW    bpb    ; таблица для трех одинаковых
          DW    bpb    ; логических устройств
          DW    bpb
```

Если ваш драйвер работает с несколькими разными по параметрам логическими устройствами, для каждого устройства необходимо подготовить свой BPB и занести его адрес в соответствующее порядковому номеру устройства место таблицы указателей на блоки BPB.

Какие действия должна выполнить функция инициализации, если выяснилось, что по тем или иным причинам установка драйвера невозможна? Например, ошибочно заданы параметры, не хватает оперативной памяти, других ресурсов или, наконец, отсутствует само обслуживаемое устройство.

Драйвер символьного устройства при этом может указать в качестве конечного адреса резидентной части программы адрес начала драйвера, т. е. адрес заголовка драйвера. Размер резидентной части при этом будет равен нулю.

Блочные драйверы дополнительно должны записать нуль в поле количества обслуживаемых логических устройств n_units.

В обоих случаях включения драйвера в состав операционной системы не произойдет.

Приведем пример простейшего драйвера символьного устройства, который обслуживает только команду инициализации. Фактически этот драйвер не рабо-

тает ни с каким физическим устройством. Он просто стирает содержимое экрана при инициализации, выводит сообщение и ожидает нажатия оператором любой клавиши. После нажатия драйвер завершает свою работу без включения себя в состав DOS.

Исходный текст драйвера:

```
.MODEL tiny
.CODE                ; Драйвер состоит из одного
                    ; сегмента кода

org 0                ; Эта строка может отсутствовать

include sysp.inc

;=====
simple PROC far      ; Драйвер - это FAR-процедура
;=====

E_O_P:              ;Метка конца программы,
                    ;устанавливаем ее в начало
                    ;драйвера, т.к. не надо
                    ;оставлять драйвер резидентно
                    ;в памяти

; Заголовок драйвера

    dd  0fffffffh    ;адрес следующего драйвера
    dw  8000h         ;байт атрибутов
    dw  dev_strategy  ;адрес процедуры стратегии
    dw  dev_interrupt ;адрес процедуры прерывания
    db  'SIMPLE_D'     ;имя устройства (дополненное
                    ;пробелами)

;=====

; Программа стратегии

dev_strategy:
    mov cs:req_seg,es
    mov cs:req_off,bx
    ret

; Здесь запоминается адрес заголовка запроса

req_seg      dw ?
req_off      dw ?

;=====

;Обработчик прерывания

dev_interrupt:
    push es        ;сохраняем регистры
    push ds
    push ax
    push bx
```



```

push cx
push dx
push si
push di
push bp

```

; Устанавливаем ES:BX на заголовок запроса

```

mov ax,cs:req_seg
mov es,ax
mov bx,cs:req_off

```

; Получаем код команды из заголовка запроса и умножаем
 ; его на два, чтобы использовать в качестве индекса
 ; таблицы адресов обработчиков команд

```

mov al,es:[bx]+2
shl al,1

```

```

sub ah,ah          ; Обнуляем AH
lea di,functions   ; DI указывает на смещение
                   ; таблицы
add di,ax           ; Добавляем смещение в таблице
jmp word ptr [di]   ; Переходим на адрес из таблицы

```

functions LABEL WORD ; Таблица функций

```

dw initialize
dw check_media
dw make_bpb
dw ioctl_in
dw input_data
dw nondestruct_in
dw input_status
dw clear_input
dw output_data
dw output_verify
dw output_status
dw clear_output
dw ioctl_out
dw Device_open
dw Device_close
dw Removable_media

```

; Выход из драйвера, если функция не поддерживается

```

check_media:
make_bpb:
ioctl_in:
nondestruct_in:
input_status:
clear_input:
output_verify:
output_status:
clear_output:
ioctl_out:
Removable_media:

```



```
Device_open:
Device_close:
output_data:
input_data:
```

```
    or     es:word ptr [bx]+3,8103h
    jmp    quit
```

```
;=====
```

```
quit:
    or     es:word ptr [bx]+3,100h
    pop    bp
    pop    di
    pop    si
    pop    dx
    pop    cx
    pop    bx
    pop    ax
    pop    ds
    pop    es
    ret
```

```
;=====
```

```
; Процедура выводит на экран строку
; символов в формате ASCIIZ
```

```
dpc proc near
```

```
    push    si
```

```
dpc_loop:
    cmp     ds:byte ptr [si],0
    jz      end_dpc
    mov     al,ds:byte ptr [si]
    @@out_ch al
    inc     si
    jmp     dpc_loop
```

```
end_dpc:
    pop     si
    ret
```

```
dpc endp
```

```
;=====
```

```
hello    db 13,10,'
          db 13,10,' *SIMPLE* (C)Frolov A., 1990 '
          db 13,10,'
          db 13,10
          db 13,10,'Hit any key...'
          db 13,10,0
```

```
;=====
```


initialize:

```
lea  ax,E_0_P      ;смещение конца программы в AX
mov  es:word ptr [bx]+14,ax  ;помещаем его в заголовок
mov  es:word ptr [bx]+16,cs  ;
```

; Стираем экран

```
mov  dh,18h
mov  dl,80h

xor  cx,cx
mov  bh,7
xor  al,al
mov  ah,6
int  10h
```

; Устанавливаем курсор в левый верхний угол экрана

```
mov  bh,0
xor  dx,dx
mov  ah,2
int  10h
```

; Выводим сообщение

```
mov  ax,cs
mov  ds,ax
mov  si,offset hello
call dpc
```

; Ожидаем нажатия на любую клавишу

```
mov  ax,0
int  16h

jmp  quit
```

simple ENDP

END simple

В программе используется макро @@out_ch, описанное в файле sysp.inc и предназначенное для вывода символов на экран дисплея. Файл sysp.inc имеется на дискете, прилагающейся к книге. Приведем текст макро @@out_ch:

```
@@out_ch  MACRO  c1,c2,c3,c4,c5,c6,c7,c8,c9,c10
            mov   ah,02h
            IRP   chr,<c1,c2,c3,c4,c5,c6,c7,c8,c9,c10>
            IFB   <chr>
            EXITM
            ENDEF
            mov   dl,chr
            int   21h
            ENDM
            ENDM
```


Текст этого драйвера транслировался при помощи ассемблера, входящего в состав QuickC 2.01. Полученный объектный модуль обрабатывался пакетным файлом:

```
link %1.obj;  
exe2bin %1.exe %1.sys
```

Вы можете также использовать макроассемблер MASM версии 5.0 или более поздней версии.

Нетрудно заметить, что процедура получения загрузочного модуля драйвера действительно похожа на процедуру получения COM-программы. Сообщение редактора об отсутствии сегмента стека следует проигнорировать, сегмента стека действительно нет и быть не может, т. к. драйвер состоит из единственного сегмента кода.

Для испытания этого и других драйверов запишите драйвер в корневой каталог системной дискеты (с которой можно загрузить операционную систему) и поместите в файл CONFIG.SYS, находящийся на этой дискете, строку:

```
DEVICE=a:\simple.sys
```

Когда ваш драйвер будет отлажен, его можно переписать на диск и подключить к файлу CONFIG.SYS, находящемуся на диске C:.

1 - Проверка замены носителя данных

Эту команду DOS выдает драйверу, когда она хочет проверить, не произошла ли замена носителя данных, например замена дискеты.

Вообще говоря, довольно трудно определить, заменил ли оператор дискету или там все еще стоит старая, с которой DOS начала работу. Наиболее достоверные результаты можно было бы получить от аппаратуры, если бы она следила за заменой дискет. Но когда аппаратура не отслеживает замены дискет, приходится анализировать метку диска или как-то иначе определять, поменялась дискета или нет. Метка диска - слабое подспорье в этом вопросе, т. к. обычно при форматировании дискет пользователь не задает метку. Однако MS-DOS 4.01 при форматировании автоматически записывает на дискету уникальный серийный номер, который можно использовать для проверки факта замены дискеты в приемном кармане дисковода.

Можно отслеживать время доступа к диску и считать, что для смены носителя нужно не менее двух секунд.

К чему может привести некорректный ответ DOS на вопрос о замене дискеты? К очень тяжелым последствиям.

Возможны три варианта ответа на вопрос о замене дискеты:

1. Да, дискета заменена.
2. Нет, дискета все та же.
3. Неизвестно, произошла замена дискеты или нет.

Если DOS получила ответ, что дискета не заменена, она продолжает работу, которой занималась раньше.

Если пришел ответ, что носитель данных заменен, DOS выдает драйверу команду с номером 2. Это запрос драйверу на построение нового BPB. Все буфера, связанные с данным устройством, при этом очищаются, и, если они не были записаны на диск, может произойти потеря информации. Осуществляется чтение каталога и FAT.

Если пришел ответ, что неизвестно, сменили ли носитель данных или нет, действия DOS зависят от наличия в данный момент непустых дисковых буферов. Если непустых буферов нет, DOS считает, что носитель сменился и ведет себя так, как это было описано раньше. Если есть непустые буфера, DOS записывает буфера на диск. При этом структура информации на новой дискете может оказаться полностью нарушенной.

Приведем формат запроса для команды проверки замены носителя:

- | | | | |
|-------|----|--------|---|
| (0) | 13 | header | - заголовок запроса |
| (+13) | 1 | media | - в этом поле драйверу передается байт-описатель среды носителя данных, с которым DOS работала раньше |
| (+14) | 1 | reply | - в это поле драйвер должен поместить ответ о факте замены среды:
1 - диск не заменялся;
0 - неизвестно;
-1 - диск был заменен |
| (+15) | 4 | vol_id | - указатель на предыдущую метку тома (если установлен бит 11 слова атрибута устройства и диск был заменен) |

Если драйвер поддерживает функцию проверки замены среды носителя данных (бит 11 слова атрибута установлен в 1) и оказалось, что произошла замена диска, драйвер должен вернуть в поле vol_id указатель на область памяти, содержащую предыдущую метку тома в формате ASCII. Если метка тома не используется драйвером, а бит 11 слова атрибутов установлен, необходимо вернуть указатель на строку "NO_NAME", закрытую двоичным нулем.

Байт-описатель среды media классифицирует используемую среду носителя данных, но делает это неоднозначно. Мы приведем характерные для этого байта параметры дисков:

- | | |
|------------|--|
| 0FFh | - 2 стороны, 8 секторов на дорожку |
| 0FEh | - 1 сторона, 8 секторов на дорожку |
| 0FDh | - 2 стороны, 8 секторов на дорожку |
| 0FCh | - 1 сторона, 9 секторов на дорожку |
| 0F9h | - 2 стороны, 15 секторов на дорожку |
| 0F8h | - жесткий диск |
| 0FEh, 0FDh | - используются восьмидюймовыми дискетами |

Более подробно все, что касается дисков, будет изложено в разделах, посвященных файловой системе.

2 - Построить блок BPB

Эта команда имеет смысл только для блочных устройств и вызывается после предыдущей команды, если произошла смена носителя данных.

Драйвер должен вернуть адрес нового блока BPB.

Формат запроса для этой команды:

(0)	13	header	- заголовок запроса
(+13)	1	media	- в этом поле драйверу передается байт-описатель среды носителя данных, с которым DOS работала раньше
(+14)	4	buf_adr	- адрес буфера обмена. Содержимое буфера при вызове драйвера зависит от установки бита 13 слова атрибутов устройства (IBM-формат). Если этот бит равен 0 (устройство формата IBM), буфер содержит первый сектор первой копии FAT. В противном случае указатель установлен на буфер свободного сектора
(+18)	4	brb_adr	- указатель на новый BPB, записывается в это поле драйвером

Операционная система MS-DOS версии 3.0 и более поздних версий включает поддержку аппаратных средств проверки смены носителя данных. Если драйвер обнаружил, что смена носителя данных произошла неправильно, в неподходящий момент времени, он может вернуть ошибку с кодом 15 (неразрешенная замена диска).

3 - IOCTL чтение

4 - Чтение

8 - Запись

9 - Запись с проверкой

12 - IOCTL запись

16 - Вывод, пока не занято

Все эти команды используются для чтения/записи информации из устройства или в устройство соответственно. Они имеют практически одинаковый формат запроса, поэтому мы их будем рассматривать вместе.

В операциях чтения или записи участвуют секторы для блочных устройств и байты для символьных устройств. Область запроса содержит указатель на буфер обмена, куда нужно поместить прочитанные данные или откуда взять данные для записи, поле количества записываемых/читаемых байт для символьных устройств или секторов для блочных.

Кроме того, драйвер должен вернуть количество действительно прочитанных или записанных байт/секторов.

Приведем формат запроса для этих команд:

- | | | | |
|-------|----|---------|--|
| (0) | 13 | header | - заголовок запроса |
| (+13) | 1 | media | - в этом поле драйверу передается байт-описатель среды носителя данных |
| (+14) | 4 | buf_adr | - адрес буфера для передачи данных |
| (+18) | 2 | count | - количество передаваемых байт для символьных устройств или секторов для блочных устройств |
| (+20) | 2 | sector | - номер начального сектора, если драйвер использует 16-битовую адресацию секторов, или -1 для 32-битовой адресации. Это поле не используется символьными драйверами |
| (+22) | 4 | vol_id | - указатель на метку тома в формате ASCIIZ. Возвращается блочным драйвером, если он выставляет ошибку 15 (неправильная смена диска). Это поле должно содержать ссылку на метку требуемого диска |
| (+26) | 4 | sect32 | - номер начального сектора, если содержимое поля sector равно -1. Первым идет старшее слово номера сектора. Если обнаружена ошибка с номером 15, в это поле записывается указатель на метку тома |

После выполнения операций чтения или записи драйвер обязательно должен записать в поле count количество действительно переданных байт для символьных устройств или секторов для блочных. В случае ошибки также требуется записать правильного значения в поле count, недостаточно только установить признак и код ошибки в слове состояния драйвера.

Для команды 9 (запись с проверкой) драйвер должен после выполнения записи проверить записанные данные. Если с консоли введена команда DOS VERIFY ON, то для дисковых устройств DOS вместо команды записи 8 использует команду записи с проверкой 9.

Драйвер может вернуть количество переданных байтов меньше запрошенного, это само по себе не является ошибкой. Иногда может получиться так, что надо выполнить запись 64 Кбайтов. Такая операция может вызвать переход за границу сегмента в буфере передачи данных. В этом случае драйвер должен проигнорировать лишние байты. Например, надо записать 10000h байтов, распределенных по секторам из буфера с адресом XXXX:0001. В этом случае драйвер должен проигнорировать последние два байта. Они будут записаны в следующий раз.

IOCTL чтение/запись - это обмен с устройством управляющей информацией. Мы будем подробно говорить об этом ниже.

Команда с кодом 16 - вывод, пока не занято, предназначена для работы с такими устройствами ввода/вывода, которые имеют свой собственный буфер, например принтеры.

Приведем исходный текст программы драйвера, использующего команды ввода и вывода. Для ввода драйвер использует клавиатуру, вывод осуществляется на экран дисплея.

```

.MODEL tiny
.CODE          ; Драйвер состоит из одного
               ; сегмента кода

org 0          ; Эта строка может отсутствовать

include sysp.inc

;=====

iodrv PROC far          ;драйвер - это FAR-процедура

;=====

; Заголовок драйвера

    dd  0fffffffh      ;адрес следующего драйвера
    dw  80000h         ;байт атрибутов
    dw  dev_strategy    ;адрес процедуры стратегии
    dw  dev_interrupt   ;адрес процедуры прерывания
    db  'IODRIVER'      ;имя устройства (дополненное
                       ; пробелами)

;=====

; Программа стратегии

dev_strategy:
    mov  cs:req_seg,es
    mov  cs:req_off,bx
    ret

; Здесь запоминается адрес заголовка запроса

req_seg      dw  ?
req_off      dw  ?

;=====

;Обработчик прерывания

dev_interrupt:
    push es          ;сохраняем регистры
    push ds
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push bp

; Устанавливаем ES:BX на заголовок запроса

    mov  ax,cs:req_seg

```



```

mov     es,ax
mov     bx,cs:req_off

; Получаем код команды из заголовка запроса и умножаем
; его на два, чтобы использовать в качестве индекса
; таблицы адресов обработчиков команд

mov     al,es:[bx]+2
shl     al,1

sub     ah,ah                ; Обнуляем AH
lea     di,functions        ; DI указывает на смещение
                                ; таблицы
add     di,ax                ; Добавляем смещение в таблице
jmp     word ptr [di]        ; Переходим на адрес из таблицы

```

functions LABEL WORD ; Таблица функций

```

dw      initialize
dw      check_media
dw      make_bpb
dw      ioctl_in
dw      input_data
dw      nondestruct_in
dw      input_status
dw      clear_input
dw      output_data
dw      output_verify
dw      output_status
dw      clear_output
dw      ioctl_out
dw      Device_open
dw      Device_close
dw      Removable_media

```

; Выход из драйвера, если функция не поддерживается

check_media:

make_bpb:

ioctl_in:

nondestruct_in:

input_status:

clear_input:

output_verify:

output_status:

clear_output:

ioctl_out:

Removable_media:

Device_open:

Device_close:

```

or      es:word ptr [bx]+3,8103h

```

```

jmp     quit

```

=====

; Обработчик команды вывода данных

output_data:


```
; Записываем в регистр CL количество  
; выводимых символов
```

```
    mov cl,es:[bx]+18  
    push cx
```

```
; Выводим сообщение о начале вывода
```

```
    mov ax,cs  
    mov ds,ax  
    mov si,offset outmsg  
    call dpc  
  
    pop cx
```

```
; Загружаем в DS:SI адрес буфера данных
```

```
    mov ax,es:[bx]+16  
    mov ds,ax  
    mov si,es:[bx]+14
```

```
; Выводим на экран символы из буфера
```

```
out_loop:  
    mov al,ds:byte ptr [si]  
    @@out_ch al  
    inc si  
    loop out_loop  
  
    jmp quit
```

```
;=====
```

```
; Сработчик команды ввода данных
```

```
input_data:
```

```
; Записываем в регистр CL количество  
; вводимых символов
```

```
    mov cl,es:[bx]+18  
    push cx
```

```
; Выводим сообщение о начале ввода
```

```
    mov ax,cs  
    mov ds,ax  
    mov si,offset inpmg  
    call dpc
```

```
; Загружаем в DS:SI адрес буфера данных
```

```
    pop cx  
    mov ax,es:[bx]+16  
    mov ds,ax  
    mov di,es:[bx]+14
```

```
; Вводим символы с клавиатуры и записываем в буфер
```

```
inp_loop:  
    mov ax,0  
    int 16h
```



```

mov  ds:byte ptr [di],al
@@out_ch al
inc di
loop inp_loop
jmp  quit

```

```

;=====
quit:

```

```

    or   es:word ptr [bx]+3,100h
    pop  bp
    pop  di
    pop  si
    pop  dx
    pop  cx
    pop  bx
    pop  ax
    pop  ds
    pop  es
    ret

```

```

;=====
; Процедура выводит на экран строку
; символов в формате ASCIIZ

```

```

dpc proc near
    push si
dpc_loop:
    cmp  ds:byte ptr [si],0
    jz   end_dpc
    mov  al,ds:byte ptr [si]
    @@out_ch al
    inc  si
    jmp  dpc_loop
end_dpc:
    pop  si
    ret
dpc endp

```

```

;=====
hello      db 13,10,'
            db 13,10,' *IODRV* (C)Frolov A., 1990
            db 13,10,'
            db 13,10,0

```

```

outmsg     DB 13,10,' Вывод на устройство IODRIVER ',0
inpmsg     DB 13,10,' Ввод с устройства IODRIVER ',0

```

```

;=====
E_O_P:      ;Метка конца программы

```

```

initialize:

```

```

    lea  ax,E_O_P      ;смещение конца программы в AX
    mov  es:word ptr [bx]+14,ax  ;помещаем его в заголовок
    mov  es:word ptr [bx]+16,cs

```



```

; Стираем экран
    mov     dh,18h
    mov     dl,80h
    xor     cx,cx
    mov     bh,7
    xor     al,al
    mov     ah,6
    int     10h

; Устанавливаем курсор в левый верхний угол экрана
    mov     bh,0
    xor     dx,dx
    mov     ah,2
    int     10h

; Выводим сообщение
    mov     ax,cs
    mov     ds,ax
    mov     si,offset hello
    call    dpc
    jmp     quit

iodrv     ENDP
                END iodrv

```

Для работы с этим драйвером можно использовать приводимую ниже программу, составленную на языке Си. Программа открывает устройство, вводит из него восемь символов, печатает введенные символы на экране и выводит их обратно на устройство:

```

#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <malloc.h>
#include <errno.h>
#include <dos.h>

int main(void);

int main(void) {
    char buf[100];
    int io_handle;
    unsigned count;

    // Открываем устройство с именем IODRIVER
    if( (io_handle = open("IODRIVER", O_RDWR)) == -1 ) {
        // Если открыть не удалось, выводим
        // код ошибки
        printf("Ошибка при открытии устройства %d",errno);
        return errno;
    }
}

```



```
// Читаем 8 байт из устройства в буфер buf
if( (count = read(io_handle, buf, 8)) == -1 ) {
    // Если при чтении произошла ошибка,
    // выводим ее код
    printf("Ошибка чтения %d", errno);
    return errno;
}

// Закрываем прочитанную строку нулем
// для последующего вывода функцией printf
buf[8]=0;

printf("\nВведена строка: %s", buf);

// Выводим только что прочитанные данные
// обратно на то же устройство
if( (count = write(io_handle, buf, 8)) == -1 ) {
    // Если при записи произошла ошибка,
    // выводим ее код
    printf("Ошибка записи %d", errno);
    return errno;
}

// Закрываем устройство
close(io_handle);
exit(0);
}
```

Эта программа служит примером того, как можно организовать взаимодействие драйвера и прикладной программы, работающей с драйвером. Позже мы приведем пример более сложного драйвера символьного устройства.

5 - Неразрушающее чтение без ожидания

Эта команда предназначена для выборки одного байта из буфера символьного устройства. Для блочных устройств эта команда не используется.

Если в буфере символьного устройства есть байты, которые могут быть прочитаны командой чтения, драйвер возвращает следующий символ, который был бы прочитан.

Если в результате проверки выяснится, что в буфере есть символы, драйвер должен установить бит 9 слова состояния устройства (занятость) в 0, в противном случае - в 1.

Эта команда используется перед выдачей команды чтения для проверки, есть ли в буфере готовые для чтения данные. Такая проверка позволяет избежать длительного ожидания готовности данных при вводе. Кроме того, команда используется для проверки наличия в буфере клавиатуры символа CTRL+BREAK.

Формат запроса для команды 5:

- | | | | |
|-------|----|--------|---|
| (0) | 13 | header | - заголовок запроса |
| (+13) | 1 | byte | - в это поле драйвер записывает извлеченный из буфера байт, который будет считан по следующей команде ввода |

6 - Проверить состояние устройства ввода

10 - Проверить состояние устройства вывода

Эти команды проверяют состояние устройства ввода и устройства вывода соответственно (только символьные устройства).

Для устройства ввода бит занятости слова состояния (бит 9) сбрасывается в 0, если буфер устройства не пуст и в нем есть готовые для чтения символы. Этот бит устанавливается драйвером в 1, когда буфер пуст и последующая команда чтения приведет к ожиданию ввода от устройства, например к ожиданию нажатия на клавишу.

Для устройства вывода бит занятости в слове состояния (бит 9) сбрасывается в 0, если нет текущих ожидающих готовности устройства запросов на вывод и последующая команда вывода может быть немедленно выполнена. Бит устанавливается в 1, если предыдущий запрос на вывод еще не обработан.

Для команд проверки состояния запрос состоит только из заголовка, область переменного формата отсутствует.

7 - Сброс буфера устройства ввода

11 - Сброс буфера устройства вывода

Эти функции заставляют драйвер сбросить все текущие запросы на ввод/вывод. Сброс буфера входного устройства используется, например, для удаления всех символов из буфера клавиатуры. После выполнения очистки буферов драйвер выставляет слово состояния и возвращает управление операционной системе.

Запрос состоит только из заголовка.

13 - Открыть устройство

14 - Закрыть устройство

Для того чтобы драйвер мог использовать эти команды, бит 11 в слове атрибутов устройства в заголовке драйвера должен быть установлен в 1.

Драйверы символьных устройств по этой команде могут посылать инициализирующие последовательности символов на устройства (например, на принтер могут посылаться команды установки типа шрифта, формата бумаги и т. д.) или устанавливать устройство в исходное состояние. Можно также обнаруживать попытки получения многократного доступа к устройству. В этом случае вторая команда открытия должна вернуть ошибку (если многократный доступ к устройству запрещен).

Драйвер блочного устройства с помощью этих команд может производить подсчет открытых файлов. Если содержимое счетчика открытых файлов для данного устройства равно 0, то открытых файлов на этом устройстве нет. Если драйвер теперь сбросит все буфера на диск, пользователь сможет заменить диск на другой. Однако если программы открывают файлы без их закрытия, то могут возникнуть ошибки при использовании этого метода.

Стандартные устройства CON, AUX, PRN всегда открыты.

Запрос для этих команд состоит только из заголовка.

15 - Проверка сменяемости диска

Эта команда используется только для тех драйверов, в слове атрибутов которых бит 11 установлен в 1. Драйвер должен сообщить DOS, возможна ли замена носителя данных. Например, замена жесткого диска обычно невозможна, хотя есть накопители со сменными жесткими дисками.

Драйвер возвращает информацию о возможности замены носителя в бите 9 слова состояния устройства. Значение этого бита, равное 0, драйвер устанавливает, если возможна замена носителя данных, в противном случае устанавливается значение, равное 1.

Запрос состоит только из заголовка.

19 - Функции общего управления вводом/выводом (IOCTL)

Команда с кодом 19 предназначена для выполнения нескольких функций и поддерживается только теми драйверами, у которых в слове атрибутов устройства установлен в 1 бит поддержки IOCTL (бит 14). Эта команда поддерживается MS-DOS версии 3.2 и более поздних версий.

Команда используется для выполнения ряда операций с дисками, доступных обычно лишь на уровне BIOS, например форматирования, чтения/записи секторов по их абсолютному номеру и т. д. Имеется стандартный интерфейс для различных типов дисков, обеспечиваемых драйвером логических дисков DRIVER.SYS.

DOS имеет специальную функцию номер 44h прерывания 21h. Эта функция имеет множество подфункций и предназначена для поддержки IOCTL. Очень скоро мы займемся этой функцией DOS, а сейчас приведем формат запроса для команды с кодом 19:

(0)	13	header	- заголовок запроса
(+13)	1	funct	- код функции команды общего IOCTL
(+14)	1	subfunc	- код подфункции для функции funct
(+15)	2	si_reg	- значение регистра SI при вызове функции 44h прерывания 21h. Эта функция DOS предназначена для управления вводом/выводом
(+17)	2	di_reg	- значение, передаваемое при вызове функции 44h прерывания 21h через регистр DI

- (+19) 4 buf - указатель на буфер данных, содержащий управляющую информацию для устройства или предназначенный для приема информации от устройства

23 - Получить активное логическое устройство

24 - Установить активное логическое устройство

Эти команды обрабатываются только теми драйверами, у которых в слове атрибутов установлен бит 6 - поддержка логических устройств. Команды используются в MS-DOS версии 3.2 и в более поздних версиях.

Команды обеспечивают метод опроса номера текущего активного логического устройства на физическом диске или установления активного логического устройства.

Приведем формат запроса:

- (0) 13 header - заголовок запроса
- (+13) 1 unit - код логического устройства, которое должно стать активным при использовании команды 24, или код активного устройства, помещаемый драйвером по команде 23
- (+14) 1 cmd - код команды
- (+15) 4 status - слово состояния
- (+19) 4 reserved - зарезервировано

По команде 23 (получить активное логическое устройство) драйвер должен поместить идентификатор устройства в поле unit, для устройства А помещается 1, для В - 2 и т. д. Если драйвер управляет единственным устройством, он должен записать в поле unit нуль.

После обзора команд перейдем к описанию функции 44h прерывания 21h. Эта функция предназначена для управления вводом/выводом и обладает широкими возможностями.

ФУНКЦИИ УПРАВЛЕНИЯ УСТРОЙСТВАМИ ВВОДА/ВЫВОДА IOCTL

В этом разделе мы рассмотрим средства управления устройствами ввода/вывода IOCTL (сокращение от Input/Output Control). Использование IOCTL - обычный метод организации связи с устройствами или получения информации об открытых файлах.

Описания будут приводиться в порядке номеров подфункций функции 44h прерывания 21h.

Для версии DOS 2.1 и более поздних версий поддерживаются подфункции с номерами от 0 до 7, для версии 3.0 дополнительно могут использоваться подфункции 8, 0Bh, для 3.1 - 9 и 0Ah, для 3.2 и более поздних версий добавляются подфункции 0Dh, 0Eh, 0Fh. Версия DOS 3.3 и более поздние версии поддерживают также подфункцию 0Ch.

INT 21: 44h 00h Получить информацию об устройстве

Вызов:

Регистр	Содержание
AH	44h
AL	00h
BX	Handle (Значение, которое операционная система возвращает при открытии файла или устройства и которое она затем использует для доступа к открытому файлу или устройству)

Возврат без ошибки:

Регистр	Содержание
CF	0
DX	Информация об устройстве

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 06h - BX содержит несуществующий или неоткрытый handle.

Функция возвращает в регистре DX информацию об устройстве, которая имеет следующий формат (для устройства):

Бит	Значение
0	Это устройство является стандартным устройством ввода
1	Стандартное устройство вывода
2	NUL-устройство
3	Часы
4	Специальное устройство
5	1 - двоичный режим работы; 0 - режим ASCII
6	0 - при чтении достигнут конец файла
7	1 - это слово информации относится к устройству (данный handle относится к устройству); 0 - слово информации относится к файлу
8-10	Зарезервировано
11	1 - Устройство поддерживает команды открытия/закрытия
12	Сетевое устройство (только для MS-DCS 3.0 и более поздних версий)
13	Устройство поддерживает вывод до состояния занятости
14	Устройство может обрабатывать управляющие строки IOCTL, посылаемые подфункциями 2, 3, 4, 5 функции 44h. Этот бит может быть только прочитан, его установка подфункцией 1 функции 44h не производится
15	Зарезервировано

Если при вызове этой подфункции регистр BX содержал handle файла, формат получаемой в регистре DX информации будет следующий:

Бит	Значение
0-5	Номер дисководов (0-A, 1-B и т.д.)
6	0 - Была запись в выходной файл
7	1 - это слово информации относится к устройству (данный handle относится к устройству); 0 - слово информации относится к файлу
8-11	Зарезервировано
12	Сетевое устройство (только для MS-DOS версии 3.0 и более поздних версий)
13-14	Зарезервировано
15	1 - Данный файл является удаленным при работе в сети (только для MS-DOS версии 3.0 и более поздних версий)

Особое внимание следует обратить на бит 5 слова информации об устройстве. Этот бит определяет режим обмена данными DOS и драйвера - двоичный или ASCII. В двоичном режиме управляющие символы CTRL+C, CTRL+P, CTRL+S, CTRL+Z интерпретируются как обычные данные.

INT 21: 44h 01h Установить информацию об устройстве

Вызов:

Регистр		Содержание
AH	44h	
AL	01h	
BX	Handle	
DH	0	
DL	Устанавливаемая информация (биты 0-7 слова информации об устройстве, описанного выше)	

Возврат без ошибки:

Регистр		Содержание
CF	0	

Возврат с ошибкой:

Регистр		Содержание
CF	1	
AX	01h - запрошена несуществующая функция; 06h - BX содержит несуществующий handle	

Эта подфункция позволяет устанавливать некоторые биты в слове информации об устройстве.

В зависимости от установки бита 5 (двоичный/ASCII) драйвер по-разному выполняет команды ввода-вывода.

В режиме ASCII обмен с драйвером символьного устройства выполняется по байтам (содержимое поля count в запросе всегда равно 1). Если вам надо обслуживать быстродействующее устройство ввода/вывода, такой посимвольный обмен может оказаться слишком медленным. В этом случае вы можете установить бит 5 в единицу и выполнять обмен в двоичном режиме. Разумеется, вам придется самостоятельно отслеживать управляющие символы.

Ниже мы приведем пример программы, которая сначала устанавливает ASCII-режим для драйвера, описанного выше, выводит на него 8 символов, затем выполняет аналогичную операцию в двоичном режиме. Драйвер при каждом обращении к нему для ввода или вывода выдает сообщение на экран. Запустив

программу (и не забыв подключить драйвер), вы увидите, что в ASCII-режиме для записи или чтения восьми символов драйвер вызывается восемь раз, а в двоичном режиме - только один раз.

```
#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <malloc.h>
#include <errno.h>
#include <dos.h>

int main(void);

union REGS inregs, outregs;
struct SREGS segregs;

int main(void) {
    char buf[100];
    int io_handle;
    unsigned count;

    // Открываем устройство с именем IODRIVER
    if( (io_handle = open("IODRIVER", O_RDWR)) == -1 ) {
        // Если открыть не удалось, выводим
        // код ошибки
        printf("Ошибка при открытии устройства %d",errno);
        return errno;
    }

    // Читаем 8 байт из устройства в буфер buf
    if( (count = read(io_handle, buf, 8)) == -1 ) {
        // Если при чтении произошла ошибка,
        // выводим ее код
        printf("Ошибка чтения %d",errno);
        return errno;
    }

    // Закрываем прочитанную строку нулем
    // для последующего вывода функцией printf
    buf[8]=0;

    printf("\n Введена строка: %s",buf);

    // Выводим только что прочитанные данные
    // обратно на то же устройство
    if( (count = write(io_handle, buf, 8)) == -1 ) {
```



```
        // Если при записи произошла ошибка,
        // выводим ее код

        printf("Ошибка записи %d",errno);
        return errno;
    }

    // Получаем информацию об устройстве

    inregs.h.ah = 0x44;
    inregs.h.al = 0;
    inregs.x.bx = io_handle;
    intdos( &inregs, &outregs );
    if(outregs.x.cflag == 1) {

        // При ошибке выводим ее код

        printf("IOCTL error %x\n",&outregs.x.ax);
        exit(-1);
    }

    // Выводим слово информации об устройстве на экран
    printf("\nDevice Information:  %04X\n", outregs.x.dx);

    // Устанавливаем в 1 бит 5 (переключаем драйвер
    // в двоичный режим обмена данными

    inregs.x.dx = (outregs.x.dx | 0x0020) & 0x00ff;

    // Устанавливаем слово информации об устройстве

    inregs.h.ah = 0x44;
    inregs.h.al = 1;
    inregs.x.bx = io_handle;
    intdos( &inregs, &outregs );
    if(outregs.x.cflag == 1) {

        // При ошибке выводим код ошибки

        printf("IOCTL error %x\n",&outregs.x.ax);
        exit(-1);
    }

    // Выводим слово информации об устройстве на экран
    printf("\nDevice Information:  %04X\n", outregs.x.dx);

    // Читаем 8 байт из устройства в буфер buf
    // Обмен теперь производится в двоичном режиме

    if( (count = read(io_handle, buf, 8)) == -1 ) {

        // Если при чтении произошла ошибка,
        // выводим ее код

        printf("Ошибка чтения %d",errno);
        return errno;
    }
}
```



```

// Закрываем прочитанную строку нулем
// для последующего вывода функцией printf
buf[8]=0;
printf("\n Введена строка: %s",buf);
// Выводим только что прочитанные данные
// обратно на то же устройство
if( (count = write(io_handle, buf, 8)) == -1 ) {
    // Если при записи произошла ошибка,
    // выводим ее код
    printf("Ошибка записи %d",errno);
    return errno;
}
// Закрываем устройство
close(io_handle);
exit(0);
}

```

INT 21: 44h 02h/03h

Чтение/запись управляющей информации для символьных устройств

Вызов:

Регистр	Содержание
AH	44h
AL	02h/03h
BX	Handle (только для устройства, а не для файла)
CX	Количество читаемых/записываемых байтов
DS:DX	Указатель на буфер, в который надо прочитать или из которого надо записать управляющую информацию

Возврат без ошибки:

Регистр	Содержание
CF	0
AX	Количество действительно прочитанных/записанных байтов

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 06h - BX содержит несуществующий handle

Эти подфункции позволяют прикладной программе обмениваться произвольной управляющей информацией с драйвером устройства. Они инициируют вызов драйвера соответственно с командами 3 и 12 для операций чтения и записи управляющей информации.

Для использования этих подфункций драйвер должен поддерживать интерфейс IOCTL. Для проверки можно использовать подфункцию 0 (чтение информации устройства). Если бит 14 установлен, драйвер поддерживает IOCTL.

INT 21: 44h 04h/05h**Чтение/запись управляющей информации для блочных устройств**

Вызов:

Регистр	Содержание
AH	44h
AL	04h/05h
BL	Номер дисковод: 0 - текущий дисковод, 1 - дисковод A и т. д.
CX	Количество читаемых/записываемых байтов
DS:DX	Указатель на буфер, в который надо прочитать или из которого надо записать управляющую информацию

Возврат без ошибки:

Регистр	Содержание
CF	0
AX	Количество действительно прочитанных/записанных байтов

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 05h - доступ к дисководу запрещен

Для использования этих подфункций драйвер должен поддерживать интерфейс IOCTL. Для проверки можно использовать подфункцию 0 (чтение информации устройства). Если бит 14 установлен, драйвер поддерживает IOCTL.

INT 21: 44h 06h/07h Получить состояние ввода/вывода

Вызов:

Регистр	Содержание
AH	44h
AL	06h/07h
BX	Handle (устройства или файла)

Возврат без ошибки:

Регистр	Содержание
CF	0
AL	Для устройства: 00h - устройство не готово к приему/передаче; 00h - устройство не готово к приему/передаче; 0FFh - устройство готово к приему/передаче Для файла: 00h - достигнут конец файла; 0FFh - конец файла не достигнут

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 05h - доступ к дисководу запрещен; 06h - BX содержит несуществующий handle; 0Dh - неправильные данные

Эти подфункции предназначены для проверки готовности устройства к вводу/выводу данных или для проверки на достижение конца файла.

INT 21: 44h 08h**Проверить возможность замены носителя данных для блочного устройства**

Вызов:

Регистр	Содержание
AH	44h
AL	08h
BL	Номер дисковода: 0 - текущий дисковод, 1 - дисковод A и т. д.

Возврат без ошибки:

Регистр	Содержание
CF	0
AX	00h - замена носителя возможна 01h - носитель данных несменный

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 0Fh - неправильный идентификатор дисковод

Эта подфункция проверяет возможность замены носителя данных. Дискета - заменяемый носитель, а жесткий диск или электронный (RAM) диск - нет.

INT 21: 44h 09h Получить информацию о том, является ли устройство локальным или удаленным (при работе в сети)

Вызов:

Регистр	Содержание
AH	44h
AL	09h
BL	Номер дисковода: 0 - текущий дисковод, 1 - дисковод A и т. д.

Возврат без ошибки:

Регистр	Содержание
CF	0
DX	Биты атрибута устройства. Если блочное устройство является локальным (расположено на рабочей станции сети Microsoft Networks), бит 12 установлен в 0, остальные биты копируются из заголовка драйвера устройства. Если устройство является удаленным (находится на сервере), устанавливается в 1 только бит 12, остальные биты зарезервированы и равны нулю

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 0Fh - неправильный идентификатор дисковода

Ошибка 01h может возникнуть, если не загружена команда SHARE.

Используя эту подфункцию, программы могут определить принадлежность дисководов локальной станции или серверу. Однако обычно работа программы не должна зависеть от расположения дисководов.

INT 21: 44h 0Ah Проверить handle на локальный/удаленный

Вызов:

Регистр	Содержание
AH	44h
AL	0Ah
BX	Handle (только для устройства, а не для файла)

Возврат без ошибки:

Регистр	Содержание
CF	0
DX	Слово атрибутов устройства, если бит 15 установлен в 1, это удаленное устройство, в противном случае это локальное устройство

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 06h - BX содержит несуществующий handle

Ошибка 01h может возникнуть, если не загружена команда SHARE.

Данная подфункция позволяет проверить, является ли открытое устройство или файл локальным или удаленным. Справедливы замечания к предыдущей подфункции о том, что работа программы не всегда должна зависеть от расположения файла или устройства.

INT 21: 44h 0Bh Установка количества повторов при обращении к файлу

Вызов:

Регистр	Содержание
AH	44h
AL	0Bh
CX	Продолжительность паузы (по умолчанию используется значение 1)

DX Число повторов (по умолчанию равно 3) до вызова программы обработки прерывания 24h - прерывания по критической ошибке ввода/вывода

Возврат без ошибки:

Регистр	Содержание
CF 0	

Возврат с ошибкой:

Регистр	Содержание
CF 1	
AX 01h	- запрошена несуществующая функция

Код ошибки 01h появляется, если задан недопустимый номер подфункции или не загружена команда SHARE.

Операции, которые могут привести к возникновению конфликтных ситуаций, автоматически повторяются несколько раз до выдачи сообщения об ошибке. Можно задавать количество повторений операций и пауз между ними. Пауза формируется при выполнении пустого цикла, и ее длительность зависит от тактовой частоты процессора на машине:

```
mov  cx, pause
loop $
```

INT 21: 44h 0Ch Переключение кодовых страниц

Вызов:

Регистр	Содержание
AH 44h	
AL 0Ch	
BX Handle	для открытого устройства
CH Код категории устройства:	
	00 - неизвестное устройство;
	01 - устройства COM1, COM2 и т. д.;
	03 - консоль CON;
	05 - устройства печати LPT1, LPT2 и т. д.
CL Код операции:	
	45h - установить число повторений операции;
	4Ah - выбор кодовой страницы;
	4Ch - начало подготовки кодовой страницы;
	4Dh - конец подготовки кодовой страницы;

5Fh - установить устройство "дисплей";
 65h - получить число повторений операции;
 6Ah - получить выбранную кодовую страницу;
 6Bh - получить подготовленный список;
 7Ah - получить параметры дисплея (ширина, длина и цвет)
 DS:DX Указатель на блок параметров

Возврат без ошибки:

Регистр	Содержание
CF 0	

Возврат с ошибкой:

Регистр	Содержание
CF 1	
AX 01h	- запрошена несуществующая функция;
	06h - BX содержит несуществующий handle

Для подготовки кодовой страницы сначала вызывают эту подфункцию с кодом операции CL=4Ch, затем должна идти серия вызовов подфункции 03h функции 44h прерывания INT 21h - запись IOCTL на символьное устройство.

Формат записываемых данных зависит от типа устройства. Драйверы DISPLAY.SYS и PRINTER.SYS, входящие в состав дистрибутива DOS, получают эти данные из файлов с расширением имени .CPI, таких, как EGA.CPI, LCD.CPI, 4201.CPI и т. д.

Блок параметров имеет различный формат для разных кодов операций:

- CL = 45h - блок параметров состоит из слова, содержащего количество повторений
- CL = 4Ah, 4Dh, 6Ah - блок параметров состоит из двух слов: длины данных (0002) и идентификатора кодовой страницы
- CL = 4Ch - в начале блока параметров расположены три слова: флаги (0000), длина остальной части блока параметров в байтах и количество кодовых страниц. За ними следуют слова кодовых страниц. Количество слов кодовых страниц в зависимости от значения MAXFONTS при установке драйвера и может достигать 12
- CL = 6Bh - первым располагается слово, содержащее длину остальной части блока в байтах, затем идет количество аппаратных кодовых страниц и слова для этих кодовых страниц, в конце расположено слово, содержащее количество подготовленных кодовых страниц и слова с подготовленными кодовыми страницами. Возвращаемый блок может иметь длину до 56 байтов

INT 21: 44h 0Dh Общее управление вводом/выводом GENERIC IOCTL

Вызов:

Регистр		Содержание
AH	44h	
AL	0Dh	
(BL		Номер дисководов: 0 - текущий дисковод, 1 - дисковод A и т. д.
CH		Код категории устройства: 08h - дисковое устройство
CL		Операции: 40h - установить параметры устройства; 60h - получить параметры устройства; 41h - записать дорожку на логическом устройстве; 61h - прочесть дорожку на логическом устройстве; 42h - форматировать дорожку на логическом устройстве; 62h - проверить дорожку на логическом устройстве
DS:DX		Указатель на блок параметров

Возврат без ошибки:

Регистр		Содержание
CF	0	

Возврат с ошибкой:

Регистр		Содержание
CF	1	
AX		Код ошибки

Общая схема использования этой подфункции:

- 1. Сохранить параметры устройства операцией 40h.
- 2. Установить нужные параметры.
- 3. Выполнить операцию ввода/вывода или форматирования.
- 4. Восстановить предыдущие параметры.

Формат блока параметров зависит от выполняемой операции:

CL = 40h/60h - получить/установить параметры устройства

Смещение	Размер	Содержимое поля
n(0)	1	Специальные функции: Бит 0: В операции 60h значение этого бита, равное 1, используется для извлечения текущего ВРВ, как если бы он был получен по команде драйвера с кодом 2 (пост-"диалог-мифи"

роить ВРВ). Значение этого бита, равное 0, означает, что надо извлечь ВРВ, используемый по умолчанию. Для операции с кодом 40h значение бита, равное 1, используется для извлечения текущего ВРВ, значение 0 приводит к использованию ВРВ, подготовленного в данном блоке параметров.

Бит 1:

Значение этого бита, равное 1, - это указание игнорировать все поля в блоке параметров, кроме поля описания физической структуры дорожки на данном устройстве.

Бит 2:

Значение этого бита, равное 1, говорит о том, что все секторы на этой дорожке имеют одинаковый размер

(+1)	1	Тип устройства, возвращаемый драйвером: 0 - флоппи-диск (5"), 320/360 Кбайт; 1 - флоппи-диск (5"), 1,2 Мбайта; 2 - флоппи-диск (3"), 720 Кбайт; 3 - 8" флоппи-диск нормальной плотности; 4 - 8" флоппи-диск двойной плотности; 5 - жесткий диск; 6 - накопитель на магнитной ленте; 7 - флоппи-диск (3"), 1,44 Мбайта
(+2)	2	Атрибуты устройства, возвращаемые драйвером. В поле используются только два младших бита. Бит 0 - признак заменяемости среды носителя данных (0 - заменяемая, 1 - не заменяемая), бит 1 - признак наличия аппаратного контроля замены дискеты (1 - контроль выполняется, 0 - контроль не выполняется). Остальные биты зарезервированы и должны содержать 0.
(+4)	2	Максимальное количество цилиндров на физическом устройстве. Это поле устанавливается драйвером.
(+6)	1	Тип среды носителя данных. Используется для устройств, поддерживающих несколько типов носителей данных, например для флоппи-дисководов на 1,2 Мбайта значение этого поля, равное 0, соответствует дискете на 1,2 Мбайта, а 1 - 360 Кбайт
(+7)	31	ВРВ для устройства. Если бит 0 поля специальных функций сброшен, то в этом поле находится новый ВРВ для устройства. Если бит 0 установлен, драйвер устройства возвращает ВРВ для всех последующих запросов на построение ВРВ
(+38)	?	Таблица разметки дорожки, имеет переменную длину.

Таблица разметки дорожки начинается с двухбайтового слова, содержащего общее количество секторов на дорожке. Затем для каждого сектора в таблице находится по два двухбайтовых слова, содержащих номер сектора (1, 2 и т. д.) и размер сектора. То есть для каждого сектора в таблице содержится два слова.

Если в поле "специальные функции" бит 2 установлен в 1, размеры всех секторов должны быть одинаковыми.

CL = 41h/61h (записать/прочитать дорожку)

<i>Смещение</i>	<i>Размер</i>	<i>Содержимое поля</i>
(0)	1	Специальные функции (это поле всегда содержит 0)
(+1)	2	Номер головки
(+3)	2	Номер дорожки
(+5)	2	Номер начального сектора: нумерация секторов, в отличие от нумерации головок и дорожек начинается с 0
(+7)	2	Общее количество секторов на дорожке, уменьшенное на единицу
(+9)	4	FAR-указатель на буфер обмена с диском, в который помещается считываемая информация или откуда берется записываемая информация

CL = 42h/62h (форматировать/проверить дорожку)

<i>Смещение</i>	<i>Размер</i>	<i>Содержимое поля</i>
(0)	1	Специальные функции. Для этой операции определен только бит 0. Перед вызовом команды значение бита, равное 0, требуется для форматирования дорожки. Если этот бит установлен в 1, то проверяется возможность использования заданного формата трека. Если после выполнения команды значение бита 0 равно 0, то поддерживается заданный формат трека и заполненную таблицу разметки дорожки можно использовать. Если значение бита 0 равно 1, то затребованный формат дорожки не поддерживается
(+1)	2	Номер головки для форматирования/проверки
(+3)	2	Номер дорожки для форматирования/проверки

INT 21: 44h 0Eh Получение информации о логическом дисковом**Вызов:**

Регистр	Содержание
AH	44h
AL	0Eh
BL	Номер дискового (0 - текущий дисковод, 1 - дисковод А и т. д.)

Возврат без ошибки:

Регистр	Содержание
CF	0
AL	Содержимое этого регистра равно 0, если данному дисководу соответствует только одно логическое устройство, или номеру текущего логического дискового (1 - А, 2 - В и т. д.)

Возврат с ошибкой:

Регистр	Содержание
CF	1
AX	01h - запрошена несуществующая функция; 0Fh - неправильный идентификатор дискового

INT 21: 44h 0Fh Установка текущего логического дискового**Вызов:**

Регистр	Содержание
AH	44h
AL	0Fh
BL	Номер дискового, который должен стать текущим (0 - текущий дисковод, 1 - дисковод А и т. д.)

Возврат без ошибки:

Регистр	Содержание
CF	0
AL	Содержимое этого регистра равно 0, если данному дисководу соответствует только одно логическое устройство, или номеру логического дискового, который будет использоваться в последующих операциях ввода/вывода (1 - А, 2 - В и т. д.)

Возврат с ошибкой:

<i>Регистр</i>		<i>Содержание</i>
CF	1	
AX	01h - запрошена несуществующая функция; 0Fh - неправильный идентификатор дисководов	

Последние две команды предназначены для работы с логическими дисковыми. На одном физическом дисковом DOS может образовать несколько логических дисков. Эти команды могут использоваться программами, например для выдачи сообщения о необходимости заменить дискету при переходе от одного логического устройства к другому, если эти устройства расположены на одном физическом накопителе.

*ГЛАВА 7***ДРАЙВЕР СИСТЕМНЫХ ЧАСОВ CLOCK\$**

Операционная система содержит драйвер системных часов. Это драйвер символического устройства, имя устройства - CLOCK\$. Символ "\$" используется для того, чтобы это имя не "конфликтовало" с именем файла CLOCK.

Характерный признак драйвера системных часов - бит 3 слова атрибутов устройства установлен в 1. Именно этот признак используется DOS для обнаружения драйвера часов, имя CLOCK\$ может быть изменено в последующих версиях операционной системы.

Для опроса содержимого часов или для установки часов DOS соответственно читает или записывает 6 байтов информации следующего формата:

<i>Смещение</i>	<i>Длина</i>	<i>Назначение</i>
(0)	2	Количество дней, прошедших после 1 января 1980 года
(+2)	1	Счетчик минут
(+3)	1	Счетчик часов
(+4)	1	Счетчик сотых долей секунды
(+5)	1	Счетчик секунд

Мы приведем пример программы, которая открывает устройство CLOCK\$, задает двоичный режим обмена информацией с ним и вводит 6 байтов, имеющих только что описанную структуру. Показания часов выводятся на экран.


```
#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <malloc.h>
#include <errno.h>
#include <dos.h>

int main(void);

union REGS inregs, outregs;
struct SREGS segregs;

int main(void) {
    int io_handle;
    unsigned count;

    struct {
        unsigned days;
        unsigned char min;
        unsigned char hours;
        unsigned char sec_per_100;
        unsigned char sec;
    } clock_buf;

    // Открываем устройство с именем CLOCK$
    if( (io_handle = open("CLOCK$", O_RDWR)) == -1 ) {
        // Если открыть устройство не удалось, выводим
        // код ошибки
        printf("Ошибка при открытии устройства %d", errno);
        return errno;
    }

    // Получаем информацию об устройстве
    inregs.h.ah = 0x44;
    inregs.h.al = 0;
    inregs.x.bx = io_handle;
    intdos( &inregs, &outregs );
    if(outregs.x.cflag == 1) {
        // При ошибке выводим ее код
        printf("IOCTL error %x\n", &outregs.x.ax);
        exit(-1);
    }

    // Устанавливаем в 1 бит 5 (переключаем драйвер
    // в двоичный режим обмена данными
    inregs.x.dx = (outregs.x.dx | 0x0020) & 0x00ff;
```



```
// Устанавливаем слово информации об устройстве
inregs.h.ah = 0x44;
inregs.h.al = 1;
inregs.x.bx = io_handle;
intdos( &inregs, &outregs );
if(outregs.x.cflag == 1) {

    // При ошибке выводим код ошибки
    printf("IOCTL error %x\n",&outregs.x.ax);
    exit(-1);
}

// Выводим слово информации об устройстве на экран
printf("\nDevice Information:  %04X\n", outregs.x.dx);

// Читаем 6 байт из устройства в буфер buf
// Обмен производится в двоичном режиме
if((count = read(io_handle, &clock_buf, 6)) == -1) {

    // Если при чтении произошла ошибка,
    // выводим ее код
    printf("Ошибка чтения %d",errno);
    return errno;
}

printf("\nПолучено от драйвера часов CLOCK$: "
       "\n"
       "\nПрошло дней после 01.01.80:  %d"
       "\nМинуты:                        %d"
       "\nЧасы:                          %d"
       "\nСекунды:                        %d"
       "\nСотые доли секунды:           %d"
       "\n",
       clock_buf.days,
       clock_buf.min,
       clock_buf.hours,
       clock_buf.sec,
       clock_buf.sec_per_100);

// Закрываем устройство
close(io_handle);
exit(0);
}
```


ГЛАВА 8

ОСОБЕННОСТИ ОТЛАДКИ ДРАЙВЕРОВ

Драйверы достаточно сложны для отладки. Это связано прежде всего с тем, что вы не сможете использовать такие отладчики, как CodeView. На этапе инициализации драйвера (при выполнении команды инициализации) загрузка операционной системы еще не завершена и воспользоваться обычным отладчиком невозможно.

Прикладная программа также не вызывает драйвер напрямую, а делает это через прерывания DOS. Отладчик CodeView не позволит вам трассировать прерывание 21h, даже если вы и сможете это сделать при помощи другого отладчика (например, отладчик Advanced Fullscreen Debug фирмы IBM позволяет трассировать операционную систему), вам придется очень долго "добираться" до программы прерывания вашего драйвера.

Малейшие ошибки в программе инициализации могут привести к невозможности завершения загрузки операционной системы. В этом случае вам придется загрузиться с дискеты и удалить строку, описывающую драйвер из файла CONFIG.SYS, затем повторить загрузку с диска.

Можно использовать специально подготовленную системную дискету, записать на нее отлаживаемый драйвер и загрузить операционную систему с дискеты. Если произойдет зависание системы, загрузите DOS с жесткого диска.

Можно порекомендовать следующую методику отладки драйвера. Программа стратегии обычно очень проста и проблем не вызывает. Для отладки программы инициализации можно подготовить специальные процедуры, отображающие на экране содержимое наиболее важных переменных и областей памяти. Такие же процедуры можно использовать и для отладки других частей драйвера.

В качестве примера приведем текст процедуры, выводящей на экран содержимое всех регистров процессора. После вывода программа ожидает нажатия любой клавиши. Текст этой процедуры следует поместить в ту часть драйвера, которая останется резидентной, тогда вы сможете вызывать ее не только при инициализации, но и при выполнении других команд.

```
; =====  
; Процедура выводит на экран содержимое  
; всех регистров и ожидает нажатия на  
; любую клавишу.  
; После возвращения из процедуры  
; все регистры восстанавливаются.
```

```
ntrace proc near
```

```
; Сохраняем в стеке регистры,  
; содержимое которых будет изменяться
```

```
    pushf  
    push ax
```



```
push bx
push cx
push dx
push ds
push bp

push cs
pop  ds

mov bp,sp

; Выводим сообщение об останове
    mov dx,offset cs:trace_msg
    @@out_str

; Выводим содержимое всех регистров
    mov ax,cs          ; cs
    call Print_word
    @@out_ch ':'
    mov ax,[bp]+14      ; ip
    call Print_word

    @@out_ch 13,10,13,10,'A','X','='
    mov ax,[bp]+10
    call Print_word

    @@out_ch ' ','B','X','='
    mov ax,[bp]+8
    call Print_word

    @@out_ch ' ','C','X','='
    mov ax,[bp]+6
    call Print_word

    @@out_ch ' ','D','X','='
    mov ax,[bp]+4
    call Print_word

    @@out_ch ' ','S','P','='
    mov ax,bp
    add ax,16
    call Print_word

    @@out_ch ' ','B','P','='
    mov ax,[bp]
    call Print_word

    @@out_ch ' ','S','I','='
    mov ax,si
    call Print_word

    @@out_ch ' ','D','I','='
    mov ax,di
    call Print_word

    @@out_ch 13,10,'D','S','='
    mov ax,[bp]+2
    call Print_word
```



```
    @@out_ch ' ', 'E', 'S', '='
    mov ax, es
    call Print_word

    @@out_ch ' ', 'S', 'S', '='
    mov ax, ss
    call Print_word

    @@out_ch ' ', 'F', '='
    mov ax, [bp]+12
    call Print_word

    lea dx, cs:hit_msg
    @@out_str

; Ожидаем нажатия на любую клавишу
    mov ax, 0
    int 16h

; Восстанавливаем содержимое регистров
    pop bp
    pop ds
    pop dx
    pop cx
    pop bx
    pop ax
    popf

    ret

trace_msg db 13, 10, '>---- BREAK ----> At address ', '$'
hit_msg db 13, 10, 'Hit any key...', '$'
ntrace endp

; =====
; Процедура выводит на экран содержимое AX
Print_word proc near
    push ax
    push bx
    push dx

    push ax
    mov cl, 8
    rol ax, cl
    call Byte_to_hex
    mov bx, dx
    @@out_ch bh
    @@out_ch bl

    pop ax
    call Byte_to_hex
    mov bx, dx
    @@out_ch bh
    @@out_ch bl
```



```

    pop dx
    pop bx
    pop ax
    ret
Print_word endp
Byte_to_hex proc near
;-----
; al - input byte
; dx - output hex
;-----
    push ds
    push cx
    push bx

    lea bx,tabl
    mov dx,cs
    mov ds,dx

    push ax
    and al,0fh
    xlat
    mov dl,al

    pop ax
    mov cl,4
    shr al,cl
    xlat
    mov dh,al

    pop bx
    pop cx
    pop ds
    ret

    tabl db '0123456789ABCDEF'
Byte_to_hex endp
end

```

Для вывода строки на экран в этой процедуре используется макро @@out_str и @@out_ch, которые определены в файле sysp.inc:

```

@@out_ch  MACRO  c1,c2,c3,c4,c5,c6,c7,c8,c9,c10
            mov    ah,02h
            IRP    chr,<c1,c2,c3,c4,c5,c6,c7,c8,c9,c10>
            IFB    <chr>
            EXITM
            ENDIF
            mov    dl,chr
            int     21h
            ENDM
            ENDM

@@out_str  MACRO
            mov    ah,9
            int     21h
            ENDM

```


Другой метод - построить макет драйвера в виде COM-программы. Используя свой любимый отладчик, вы сможете проверить работу большинства входящих в драйвер модулей.

Если у вас есть отладчик Advanced Fullscreen Debug, можно использовать его способность оставаться резидентным и вызываться по нажатию комбинации клавиш CTRL+ESC.

В интересующее вас место драйвера поместите вызов прерывания 16h, ожидающий ввода с клавиатуры, например:

```
push ax
mov ax, 0
int 16h
pop ax
```

Можно сохранить в стеке и регистр флагов, если его изменение нежелательно.

После того как драйвер "повиснет" на ожидании ввода, активизируйте отладчик, нажав комбинацию клавиш CTRL+ESC. Вы окажетесь в теле обработчика прерывания 16h. Выполняя программу по шагам, довольно скоро вы достигнете выхода из этого обработчика - команды IRET. После выполнения команды IRET управление будет передано команде, следующей за командой int 16h. Эта команда (в приведенном примере - pop ax) принадлежит вашему драйверу!

Используя известное теперь значение адреса заголовка запроса (регистры ES:BX), можно определить, какая команда выполняется драйвером и просмотреть сам запрос.

Выполнив все необходимые отладочные действия, запустите программу на выполнение без отладки. В нужный момент времени вы снова сможете вызвать отладчик тем же способом.

Теоретически возможно создать такой отладчик, который сам оформлен в виде драйвера, и подключить его в файле CONFIG.SYS до проверяемого драйвера. Тогда весь процесс загрузки и инициализации будет производиться под контролем этого драйвера-отладчика.

Программа-драйвер использует системный стек, имеющий довольно небольшой размер. Поэтому при необходимости организуйте свой стек в области памяти, принадлежащей драйверу.

Очень важно, чтобы драйвер перед началом работы сохранил содержимое всех регистров, включая регистр флагов, а перед возвратом управления операционной системе восстановил старое содержимое регистров.

Критичные участки драйвера должны выполняться с замаскированными прерываниями.

ГЛАВА 9

ПРИМЕР ДРАЙВЕРА СИМВОЛЬНОГО УСТРОЙСТВА

Приведем пример драйвера символьного устройства, который вы можете взять в качестве прототипа своей разработки. Этот драйвер выполняет следующие действия:

- принимает и анализирует строку параметров из команды "DEVICE=" файла CONFIG.SYS, преобразует параметры из символьной формы в двоичную и проверяет их на корректность
- если параметры заданы неправильно, в процессе инициализации выводится сообщение и драйвер не подключается к операционной системе
- драйвер переназначает одно прерывание, номер которого задается в строке параметров
- обработчик переназначенного прерывания моделирует выполнение функций ввода/вывода и выполняет обработку неправильной функции
- демонстрируется использование функций IOCTL и ввода/вывода, ввод данных драйвер производит с клавиатуры, вывод осуществляет на экран дисплея

Приведем полный текст драйвера.

```
;
; Демонстрационный драйвер символьного устройства
;
; Copyright (C)Frolov A., 1990
;

.MODEL tiny
.CODE                ; Драйвер состоит из одного
                    ; сегмента кода

org 0                ; Эта строка может отсутствовать

include sysp.inc

devdrv PROC far

; Заголовок драйвера

    dd  0ffffffffh    ; адрес следующего драйвера
    dw  0C800h        ; байт атрибутов
    dw  dev_strategy  ; адрес процедуры стратегии
    dw  dev_interrupt  ; адрес процедуры прерывания
    db  'DEVDRIVR'    ; имя устройства (дополненное
                    ; пробелами)
```



```
;=====
; Программа стратегии
dev_strategy:
    mov     cs:req_seg,es
    mov     cs:req_off,bx
    ret

; Здесь запоминается адрес заголовка запроса
req_seg      dw ?
req_off      dw ?
;=====

;Обработчик прерывания
dev_interrupt:
    push es      ;сохраняем регистры
    push ds
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push bp

; Устанавливаем ES:BX на заголовок запроса
    mov     ax,cs:req_seg
    mov     es,ax
    mov     bx,cs:req_off

; Получаем код команды из заголовка запроса и умножаем
; его на два, чтобы использовать в качестве индекса
; таблицы адресов обработчиков команд
    mov     al,es:[bx]+2
    shl     al,1

    sub     ah,ah      ; Обнуляем AH
    lea     di,functions ; DI указывает на смещение
                        ; таблицы
    add     di,ax      ; Добавляем смещение в таблице
    jmp     word ptr [di] ; Переходим на адрес из таблицы

functions     LABEL WORD ; Таблица функций
    dw     initialize
    dw     check_media
    dw     make_bpb
    dw     ioctl_in
    dw     input_data
    dw     nondestruct_in
    dw     input_status
    dw     clear_input
```



```
dw    output_data
dw    output_verify
dw    output_status
dw    clear_output
dw    ioctl_out
dw    Device_open
dw    Device_close
dw    Removable_media
dw    reserved
dw    reserved
dw    reserved
dw    generic_ioctl
```

; Выход из драйвера, если функция не поддерживается

```
check_media:
make_bpb:
clear_input:
output_verify:
clear_output:
Removable_media:
reserved:
generic_ioctl:
```

; Выводим сообщение о вызове
; неподдерживаемой драйвером команды

```
    mov ax,cs
    mov ds,ax
    mov si,offset errmsg
    call dpc
```

; Ожидаем нажатия на любую клавишу

```
    mov ax,0
    int 16h
```

; Устанавливаем признак ошибки

```
    or    es:word ptr [bx]+3,8103h
    jmp   quit
```

;=====

```
nondestruct_in:
```

; Выводим сообщение о начале неразрушающего ввода

```
    mov ax,cs
    mov ds,ax
    mov si,offset inpmg_nd
    call dpc
```

; Вводим символ с клавиатуры и помещаем
; его в область запроса

```
    mov ax,0
    int 16h
```



```
    mov  BYTE PTR es:[bx]+0dh,al
    jmp  quit

;=====

input_status:

; Выводим сообщение о вызове команды
; проверки состояния ввода
    mov  ax,cs
    mov  ds,ax
    mov  si,offset statmsg_i
    call dpc

; Устанавливаем признак "Занято", т.к.
; последующая команда чтения приведет к ожиданию
; нажатия на клавишу (буферизация не используется)
    or   es:word ptr [bx]+3,0200h
    jmp  quit

;=====

output_status:

; Выводим сообщение о вызове команды
; проверки состояния вывода
    mov  ax,cs
    mov  ds,ax
    mov  si,offset statmsg_o
    call dpc

; Бит занятости не устанавливаем, т.к.
; считаем, что консоль доступна для вывода
    or   es:word ptr [bx]+3,0000h
    jmp  quit

;=====

; Обработчик команды вывода данных
output_data:

; Записываем в регистр CL количество
; выводимых символов
    mov  cl,es:[bx]+18
    push cx

; Выводим сообщение о начале вывода
    mov  ax,cs
    mov  ds,ax
    mov  si,offset outmsg
    call dpc

    pop  cx
```


; Загружаем в DS:SI адрес буфера данных

```
mov ax,es:[bx]+16
mov ds,ax
mov si,es:[bx]+14
```

; Выводим на экран символы из буфера

```
out_loop:
    mov al,ds:byte ptr [si]
    @@out_ch al
    inc si
    loop out_loop
    jmp quit
```

=====

; Обработчик команды ввода данных

input_data:

; Записываем в регистр CL количество
; ВВОДИМЫХ СИМВОЛОВ

```
mov cl,es:[bx]+18
push cx
```

; Выводим сообщение о начале ввода

```
mov ax,cs
mov ds,ax
mov si,offset inpmsg
call dpc
```

; Загружаем в DS:SI адрес буфера данных

```
pop cx
mov ax,es:[bx]+16
mov ds,ax
mov di,es:[bx]+14
```

; Вводим символы с клавиатуры и записываем в буфер

```
inp_loop:
    mov ax,0
    int 16h
    mov ds:byte ptr [di],al
    @@out_ch al
    inc di
    loop inp_loop
    jmp quit
```

=====

; Обработчик команды вывода данных IOCTL

ioctl_out:


```
; Записываем в регистр CL количество
; выводимых символов
    mov cl,es:[bx]+18

; Загружаем в DS:SI адрес буфера данных
    mov ax,es:[bx]+16
    mov ds,ax
    mov si,es:[bx]+14

; Выводим на экран символы из буфера
ioctl_out_loop:
    mov al,ds:byte ptr [si]
    @@out_ch al
    inc si
    loop ioctl_out_loop

    jmp quit

;=====

; Обработчик команды ввода данных IOCTL
ioctl_in:

; Записываем в регистр CL количество
; вводимых символов
    mov cl,es:[bx]+18

; Загружаем в DS:SI адрес буфера данных
    mov ax,es:[bx]+16
    mov ds,ax
    mov di,es:[bx]+14

; Вводим символы с клавиатуры и записываем в буфер
ioctl_inp_loop:
    mov ax,0
    int 16h
    mov ds:byte ptr [di],al
    @@out_ch al
    inc di
    loop ioctl_inp_loop

    jmp quit

;=====

Device_open:

; Выводим сообщение об открытии устройства
    mov ax,cs
    mov ds,ax
    mov si,offset openmsg
```



```
call dpc
```

```
jmp quit
```

```
;=====
```

```
Device_close:
```

```
; Выводим сообщение о закрытии устройства
```

```
mov ax,cs
```

```
mov ds,ax
```

```
mov si,offset closemsg
```

```
call dpc
```

```
jmp quit
```

```
;=====
```

```
quit:
```

```
or es:word ptr [bx]+3,100h
```

```
pop bp
```

```
pop di
```

```
pop si
```

```
pop dx
```

```
pop cx
```

```
pop bx
```

```
pop ax
```

```
pop ds
```

```
pop es
```

```
ret
```

```
;=====
```

```
parm_off dw ? ; Смещение строки параметров
```

```
parm_seg dw ? ; Сегмент строки параметров
```

```
pc_type dw ? ; Область памяти для сохранения
```

```
int_num dw ? ; значений параметров
```

```
out_port dw ?
```

```
inp_port dw ?
```

```
ctrl_inp_port dw ?
```

```
ctrl_out_port dw ?
```

```
;=====
```

```
; **
```

```
; .Name            INTERRUPT
```

```
; .Title           Обработчик прерывания
```

```
; .Synopsis
```

```
; -
```

```
; int <NN>
```

```
; Вход: NN - Номер прерывания, заданный в файле CONFIG.SYS
```

```
; AH - Номер выполняемой функции:
```

```
;        0 - операция записи;
```

```
;        1 - операция чтения
```

```
; BH - Адрес (0...7Fh)
```

```
; BL - Данные для записи (0...FFh)
```



```
; Выход: BL - Прочитанные данные
;-
;.Description
;-
; Прерывание вызывается командой INT <NN> с параметрами:
;
; Вход:  NN - Номер прерывания, заданный в файле CONFIG.SYS
;        AH - Номер выполняемой функции:
;            0 - операция записи;
;            1 - операция чтения
;        BH - Адрес (0...7Fh)
;        BL - Данные для записи (0...FFh)
; Выход: BL - Прочитанные данные
;-
;.Returns
; BL - Прочитанные данные
;.Version 1.00 (c)Copyright Frolov A., 1990
;-
;..
```

```
interrupt:
    push ax
    push cx
    push dx
    push bp
    push si
    push di
    push ds
    push es

    cmp ah,0      ; команда записи
    jz int_write
    cmp ah,1      ; команда чтения
    jz int_read

; Обработка неизвестной команды
    @@out_ch 13,10,'?','?','?',13,10

; Устанавливаем признак ошибки
    mov ax,0ffffh
    jmp int_exit

int_write:
; Выводим сообщение о приходе прерывания,
; предназначенного для записи
    @@out_ch 13,10,'W','R','I','T','E',13,10
    mov ax,0
    jmp int_exit

int_read:
; Выводим сообщение о приходе прерывания,
; предназначенного для чтения
    @@out_ch 13,10,'R','E','A','D',13,10
```


; Имитация чтения, всегда возвращается значение 55h

```
mov bl,55h
mov ax,0
jmp int_exit
```

```
int_exit:
    pop es
    pop ds
    pop di
    pop si
    pop bp
    pop dx
    pop cx
    pop ax
```

```
iret
```

; Процедура выводит на экран строку
; символов в формате ASCIIZ

```
dpc proc near
    push si
dpc_loop:
    cmp ds:byte ptr [si],0
    jz end_dpc
    mov al,ds:byte ptr [si]
    @@out_ch al
    inc si
    jmp dpc_loop
```

```
end_dpc:
    pop si
    ret
```

```
dpc endp
```

```
hello db 13,10,'
       db 13,10,' *DEVDRV* (C)Frolov A., 1992
       db 13,10,'
       db 13,10,0
```

```
outmsg  DB 13,10,' Вывод на устройство DEVDRVR ',0
inpmsg  DB 13,10,' Ввод с устройства DEVDRVR ',0
```

```
openmsg db 13,10,' Открываем DEVDRVR ',0
closemsg db 13,10,' Закрываем DEVDRVR ',0
```

```
inpmsg_nd DB 13,10
           DB ' ND-ввод с устройства DEVDRVR ',0
```

```
statmsg_i DB 13,10
           DB ' Чтение состояния ввода DEVDRVR ',0
```

```
statmsg_o DB 13,10
           DB ' Чтение состояния вывода DEVDRVR ',0
```



```
errmsg      DB 13,10
            DB '■■■■ Команда не поддерживается DEVDRIVR ■■■■',0
;=====
```

```
E_O_P:      ;Метка конца программы
```

```
initialize:
```

```
    lea ax,E_O_P      ;смещение конца программы в AX
    mov es:word ptr [bx]+14,ax ;помещаем его в заголовок
    mov es:word ptr [bx]+16,cs ;
```

```
    mov ax,es:word ptr [bx]+18 ;смещение строки
                                ; параметров
```

```
    mov cs:parm_off,ax
    mov ax,es:word ptr [bx]+20 ;сегмент строки
                                ; параметров
```

```
    mov cs:parm_seg,ax
```

```
; Стираем экран
```

```
    mov dh,18h
    mov dl,80h
```

```
    xor cx,cx
    mov bh,7
    xor al,al
    mov ah,6
    int 10h
```

```
; Устанавливаем курсор в левый верхний угол экрана
```

```
    mov bh,0
    xor dx,dx
    mov ah,2
    int 10h
```

```
; Выводим сообщение
```

```
    mov ax,cs
    mov ds,ax
    mov si,offset hello
    call dpc
```

```
; Раскодируем строку параметров и проверяем
; корректность заданных параметров
```

```
    push cs
    pop ds
```

```
    mov bx, cs:parm_off ; ES:BX - адрес строки
                        ; параметров
```

```
    mov ax, cs:parm_seg
    mov es, ax
```

```
; Адрес начала области параметров
```

```
    mov bp, OFFSET cs:pc_type
```



```
; Анализируем параметры
```

```
    call    parm
    jc      parm_errors
```

```
; Устанавливаем вектор прерывания с номером,
; заданным в строке параметров
```

```
    push cs
    pop ds
    mov dx,OFFSET cs:interrupt
    mov ax,cs:int_num
    mov ah,25h
    int 21h

    jmp quit
```

```
parm_errors:
```

```
; Если параметры заданы с ошибкой,
; установку драйвера не производим.
```

```
    mov ax,cs:req_seg      ;ES:BX указывают на заголовок
                           ; запроса
    mov es,ax
    mov bx,cs:req_off

    lea ax,devdrv          ;смещение начала драйвера
    mov es:word ptr [bx]+14,ax ;помещаем его в заголовок
    mov es:word ptr [bx]+16,cs ;

    jmp quit
```

```
;=====
```

```
parm proc near
```

```
    xor     si, si ; индекс в строке параметров
```

```
next_chr: mov     al, BYTE PTR es:[bx][si]
            cmp     al, 0ah ; проверки на конец
            je      parm_br ; строки параметров
            cmp     al, 0dh
            je      parm_br
            cmp     al, 0h
            jz      parm_br
```

```
; Копируем очередной байт строки параметров в буфер
```

```
    mov     BYTE PTR cs:parm_buffer[si], al
    inc     si
    jmp     next_chr
```

```
; Закрываем скопированную строку параметров нулем
```

```
parm_br:  mov     BYTE PTR cs:parm_buffer[si], 0
```


; Подготавливаем регистры для вызова программы
; анализа параметров и проверяем правильность
; заданных параметров

```
mov     dx, OFFSET sep
mov     cl, 6
mov     bx, OFFSET cs:parm_buffer

call    get_parm
jc      err_msg

mov     ax, cs:pc_type
cmp     ax, 0
jz      model_is_valid
cmp     ax, 1
jz      model_is_valid

jmp     err_msg
```

model_is_valid:

; Если параметры заданы правильно,
; выводим их значения на экран

```
mov     si, OFFSET msg1
call    dpc
mov     ax, cs:pc_type
call    print_word

mov     si, OFFSET msg2
call    dpc
mov     ax, cs:int_num
call    print_word

mov     si, OFFSET msg3
call    dpc
mov     ax, cs:out_port
call    print_word

mov     si, OFFSET msg4
call    dpc
mov     ax, cs:inp_port
call    print_word

mov     si, OFFSET msg41
call    dpc
mov     ax, cs:ctrl_inp_port
call    print_word

mov     si, OFFSET msg42
call    dpc
mov     ax, cs:ctrl_out_port
call    print_word

@@out_ch 13, 10, 13, 10
clc
jmp     end_of_parm
```

err_msg:


```
; Если были ошибки в параметрах, выводим
; сообщение об ошибке,
; самую ошибочную строку параметров
; и ожидаем нажатия на любую клавишу.
; На выходе устанавливаем флаг CARRY
```

```
    mov     si, OFFSET msg5
    call    dpc
```

```
    mov     ds, cs:parm_seg
    mov     si, cs:parm_off
    call    dpline
```

```
    mov     ax, cs
    mov     ds, ax
    mov     si, OFFSET msg6
    call    dpc
```

```
    mov     ax, 0
    int     16h
    stc
```

```
end_of_parm:  ret
```

```
parm_buffer db  100 DUP (?)
```

```
sep         db      " ", 0
```

```
msg1        db      13, 10, "Personal Computer Type      ", 0
```

```
msg2        db      13, 10, "Used Interrupt Number     ", 0
```

```
msg3        db      13, 10, "Device Input Port      ", 0
```

```
msg4        db      13, 10, "Device Output Port     ", 0
```

```
msg41       db      13, 10, "Device Inp Control Port ", 0
```

```
msg42       db      13, 10, "Device Out Control Port ", 0
```

```
msg5        db      13, 10, "Driver Parameters Error!", 13, 10, 0
```

```
msg6        db      13, 10, 13, 10, " Press any key...", 13, 10, 0
```

```
parm endp
```

```
...
```

```
; .Name      get_parm
```

```
; .Title     Разбор строки параметров.
```

```
; .Synopsis
```

```
;           ds:bx  -  исходная строка, закрытая нулем
```

```
;           ds:dx  -  строка из сепараторов, закрытая
;                   нулем
```

```
;           ds:bp  -  буфер параметров
```

```
;           cx    -  число параметров
```

```
; .Description
```

```
;           Исходная строка ds:bx, разделенная
;           сепараторами ds:dx, состоящая из cx
;           параметров и начинающаяся с полного пути
;           программы, разбирается на слова, параметры -
;           шестнадцатеричные цифры. Двоичные слова,
```



```

;           соответствующие параметрам, последовательно
;           заносятся в буфер параметров ds.bp.
;.Returns   В случае ошибки устанавливается флаг
;           переноса.
;.Version 1.00 (c)Copyright Frolov G., 1990
;..**

get_parm proc near

    push    bx
    push    cx
    push    ax
    push    si

    xor     ch, ch
    xor     ax, ax
    mov     si, ax

    call    strtoc
    jc      parm_end

parm_loop:
    mov     ax, 22h
    call    strtoc
    jc      parm_end
    call    hex_to_bin
    jc      parm_end
    mov     ds:[bp][si], ax
    inc     si
    inc     si
    loop    parm_loop

parm_end:
    pop     si
    pop     ax
    pop     cx
    pop     bx

    ret

get_parm endp

;..**
;.Name      strtoc
;.Title     Выделяет очередное слово из строки.
;.Synopsis
;
;   1) при первом обращении к процедуре:
;       Вход:
;           ax = 0
;           ds:bx - исходная строка, закрытая нулем
;           ds:dx - строка из сепараторов, закрытая
;                   нулем
;       Выход:
;           ds:bx - подстрока до первого разделителя,
;                   закрытая нулем

```



```

;      2) при последующих обращениях
;      Вход:
;          ax != 0
;          ds:dx  - строка из сепараторов, закрытая
;                  нулем
;      Выход:
;          ds:bx  - подстрока до следующего
;                  разделителя, закрытая нулем
;
;.Description      При первом вызове выделяет из строки первое
;                  слово до разделителя. При повторном вызове
;                  возвращает указатель на следующее слово в
;                  исходной строке. Если все слова из строки
;                  уже выделены, устанавливается флаг
;                  переноса.
;
;.Version 1.00 (c)Copyright Frolov G., 1990
; **

```

```

strtoc proc near

```

```

        push    bp
        push    di

        mov     space, 0

        cmp     ax, 0
        jz      first

        mov     bx, cs:ds1_off
        mov     ax, cs:ds1_seg
        mov     ds, ax

first:   cmp     BYTE PTR ds:[bx], 0
        jz      error

        mov     bp, bx
str_begin: mov    di, dx

compe:   mov     ah, BYTE PTR ds:[bp]
        cmp     ah, 0
        jz      lab
        cmp     BYTE PTR ds:[di], ah
        jne     next
        mov     BYTE PTR ds:[bp], 0
        inc     bp
        inc     BYTE PTR cs:space
        jmp     str_begin

lab:     mov     WORD PTR cs:ds1_off, bp
        mov     ax, ds
        mov     WORD PTR cs:ds1_seg, ax
        jmp     end_proc

next:    inc     di
        cmp     BYTE PTR ds:[di], 0
        jnz     compe

```



```

        cmp     BYTE PTR cs:space, 0
        jnz     lab
        inc     bp
        jmp     str_begin
error:   stc
end_proc: pop     di
        pop     bp
        ret

```

```

ds1_off  dw      ?
ds1_seg  dw      ?
space    db      ?

```

```
strtoc endp
```

```

; **
;
; .Name          hex_to_bin
; .Title         Преобразует  hex строку в двоичное число.
; .Synopsis
;
;               Вход:
;               ds:bx  -  исходная строка, закрытая нулем
;               Выход:
;               ax    -  соответствующее строке число
;
; .Description   Преобразует строку из ascii символов в hex
;               форме в ее 16-битовый двоичный эквивалент.
;               В случае переполнения или если строка
;               содержит символы, не равные 0..9, A..F, a..f,
;               устанавливается флаг переноса.
;
; .Version 1.00 (c)Copyright Frolov G., 1990
; **

```

```
hex_to_bin PROC NEAR
```

```

        push    bp
        push    si
        push    dx

        xor     ax, ax
        mov     bp, bx

begin:   cmp     BYTE PTR ds:[bp], 0h
        jz      end_pro
        call    hex
        jc      h_error

        mov     si, 10h
        xor     dx, dx
        mul     si
        cmp     dx, 0
        jnz     h_error

```



```

        add    ax, bx
        inc    bp
        jmp    begin

h_error: stc
end_pro:
        pop    dx
        pop    si
        pop    bp

        ret
hex_to_bin endp

hex proc near
        xor     bh, bh
        mov     bl, BYTE PTR ds:[bp]

        cmp     bl, '0'
        jb      hex_error
        cmp     bl, '9'
        ja      next_big
        and     bx, 0fh
        jmp     ok

next_big: cmp     bl, 'A'
        jb      hex_error
        cmp     bl, 'F'
        ja      next_small
        sub     bl, 37h
        jmp     ok

next_small: cmp     bl, 'a'
        jb      hex_error
        cmp     bl, 'f'
        ja      hex_error
        sub     bl, 57h
        jmp     ok

hex_error: stc
ok:        ret
hex endp

```

```

; **
;
; .Name          dec_to_bin
; .Title         Преобразует dec строку в двоичное число.
; .Synopsis
;
;      Вход:
;      ds:bx  -  исходная строка, закрытая нулем
;      Выход:
;      ax    -  соответствующее строке число
;
; .Description   Преобразует строку из ascii символов в dec
;                форме в ее 16-битовый двоичный эквивалент.
;                В случае переполнения или если строка содержит

```



```

;           символы, не равные 0..9, устанавливается флаг
;           переноса.
;..Version 1.00 (c)Copyright Frolov G., 1990
;..**
dec_to_bin proc near
    push    bp
    push    si
    push    dx

    xor     ax, ax
    mov     bp, bx

d_begin:   cmp     BYTE PTR ds:[bp], 0h
    jz      d_end_pro

    cmp     BYTE PTR ds:[bp], '0'
    jb      d_error
    cmp     BYTE PTR ds:[bp], '9'
    ja      d_error
    mov     si, 10
    xor     dx, dx
    mul     si
    cmp     dx, 0
    jnz     d_error
    mov     bl, BYTE PTR ds:[bp]
    and     bx, 0fh
    add     ax, bx
    inc     bp
    jmp     d_begin

d_error:   stc

d_end_pro: pop     dx
    pop     si
    pop     bp

    ret

dec_to_bin endp

print_word proc near
;-----
    push    ax
    push    bx
    push    dx

;
    push    ax
    mov     cl, 8
    rol     ax, cl
    call    byte_to_hex
    mov     bx, dx
    @@out_ch bh
    @@out_ch bl

;
    pop     ax
    call    byte_to_hex

```



```
        mov bx,dx
        @@out_ch bh
        @@out_ch bl
;
        pop dx
        pop bx
        pop ax
        ret
print_word endp

byte_to_hex proc near
;-----
; al - input byte
; dx - output hex
;-----
        push ds
        push cx
        push bx
;
        lea bx,tabl
        mov dx,cs
        mov ds,dx
;
        push ax
        and al,0fh
        xlat
        mov dl,al
;
        pop ax
        mov cl,4
        shr al,cl
        xlat
        mov dh,al
;
        pop bx
        pop cx
        pop ds
        ret
;
tabl db '0123456789ABCDEF'
byte_to_hex endp

;-----
dpline proc near
        push si
dpline_loop:
        cmp ds:byte ptr [si],0dh
        jz  end_dpline
        cmp ds:byte ptr [si],0ah
        jz  end_dpline

        mov al,ds:byte ptr [si]
        @@out_ch al
        inc si
        jmp dpline_loop

end_dpline:
```



```
    pop si
    ret
dpline endp

devdrv ENDP

    END devdrv
```

Для работы с этим драйвером и демонстрации его основных возможностей мы подготовили следующую программу:

```
// Данная программа использует прерывание 80h,
// которое устанавливается демонстрационным
// драйвером. Для правильной установки файл
// CONFIG.SYS должен содержать, например,
// такую строку для подключения драйвера:
//
// device=e:\sysprg\devdrv.sys 1 80 378 379 37a 37a
//
// Число 80 означает номер используемого прерывания.
```

```
#include <io.h>
#include <conio.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys\types.h>
#include <sys\stat.h>
#include <malloc.h>
#include <errno.h>
#include <dos.h>
```

```
int main(void);
```

```
union REGS inregs, outregs;
struct SREGS segregs;
```

```
int main(void) {
```

```
    char buf[100], ch;
    int io_handle;
    unsigned count;
```

```
// Открываем устройство с именем DEVDRIVER
```

```
if( (io_handle = open("DEVDRIVER", O_RDWR)) == -1 ) {
```

```
    // Если открыть устройство не удалось, выводим
    // код ошибки
```

```
    printf("Ошибка при открытии устройства %d", errno);
    return errno;
```

```
}
```

```
// Читаем 8 байт из устройства в буфер buf
```

```
printf("\nВведите 8 символов с клавиатуры\n");
```



```
if( (count = read(io_handle, buf, 8)) == -1 ) {
    // Если при чтении произошла ошибка,
    // выводим ее код
    printf("Ошибка чтения %d",errno);
    return errno;
}

// Закрываем прочитанную строку нулем
// для последующего вывода функцией printf
buf[8]=0;

printf("\nВведена строка: %s",buf);

// Выводим только что прочитанные данные
// обратно на то же устройство
if( (count = write(io_handle, buf, 8)) == -1 ) {
    // Если при записи произошла ошибка,
    // выводим ее код
    printf("Ошибка записи %d",errno);
    return errno;
}

// Вводим строку IOCTL из устройства
printf("\nВведите строку IOCTL (8 символов): ");

inregs.h.ah = 0x44;
inregs.h.al = 2;
inregs.x.bx = io_handle;
inregs.x.dx = (unsigned)buf;
inregs.x.cx = 8;
intdos( &inregs, &outregs );
if(outregs.x.cflag == 1) {
    // При ошибке выводим код ошибки
    printf("IOCTL error %x\n",&outregs.x.ax);
    exit(-1);
}
buf[8]=0;
printf("\nВведена строка IOCTL: %s",buf);

// Выводим строку IOCTL на устройства из buf
printf("\nВыведена строка IOCTL: ");

inregs.h.ah = 0x44;
inregs.h.al = 3;
inregs.x.bx = io_handle;
inregs.x.dx = (unsigned)buf;
inregs.x.cx = 8;
intdos( &inregs, &outregs );
if(outregs.x.cflag == 1) {
```



```
// При ошибке выводим код ошибки
printf("IOCTL error %x\n",&outregs.x.ah);
exit(-1);
}

printf("\n\n\nПроверяем вызов прерывания."
      "\n"
      "\nНажмите любую клавишу...\n\n");
getch();

printf("\nКоманда записи:\n");

inregs.h.ah = 0x0; /* WRITE */
inregs.h.bh = 0x777;
inregs.h.bl = 0x13;
int86( 0x80, &inregs, &outregs );

printf("\nКоманда чтения:\n");

inregs.h.ah = 0x1; /* READ */
inregs.h.bh = 0x776;
int86( 0x80, &inregs, &outregs );
ch=outregs.h.bl;

printf("Полученное значение: %x\n",ch);

printf("\nНеизвестная команда:\n");

inregs.h.ah = 0x2; /* ??? */
int86( 0x80, &inregs, &outregs );

// Закрываем устройство
close(io_handle);

exit(0);
}
```

ГЛАВА 10

ПРИМЕР ДРАЙВЕРА БЛОЧНОГО УСТРОЙСТВА

Приведем пример драйвера "электронного" диска, расположенного в основной (не расширенной или дополнительной) памяти компьютера. Этот драйвер предназначен, разумеется, не для замены поставляющегося стандартно RAMDRIVE.SYS, однако на его примере можно увидеть, как устроены драйверы блочных устройств.

И если когда-нибудь вам потребуется использовать диски ЭВМ серии ЕС в качестве винчестера персонального компьютера, то, разобравшись в том, как ра-

ботает приведенный ниже драйвер, вы сможете самостоятельно приспособить его для такой задачи.

```

;
; Драйвер электронного диска,
; использует основную память компьютера
;

        .MODEL tiny
        .CODE          ; Драйвер состоит из одного
                        ; сегмента кода

        org 0           ; Эта строка может отсутствовать

        include sysp.inc

ram PROC far

;=====

; Заголовок драйвера

        dd  0fffffffh    ; адрес следующего драйвера
        dw  2000h         ; байт атрибутов
        dw  dev_strategy ; адрес процедуры стратегии
        dw  dev_interrupt ; адрес процедуры прерывания
        db  1
        db  7 dup(?)

; Блок BPB для электронного диска

bpb     equ $

        dw  512           ; количество байтов в секторе
        db  1             ; количество секторов в кластере
        dw  1             ; количество зарезервированных секторов
        db  2             ; количество копий FAT
        dw  64            ; макс. количество файлов в корневом каталоге
        dw  360           ; общее количество секторов
        db  0fch          ; описатель среды носителя данных
        dw  2             ; количество секторов на одну копию FAT

bpb_ptr dw bpb           ; указатель на блок BPB

; Область локальных переменных драйвера

total   dw  ?            ; количество секторов
verify  db  0            ; флаг проверки при записи
start_sec dw  0          ; номер начального сектора
vdisk_ptr dw  0          ; сегмент начала участка памяти,
                        ; в котором расположен диск

user_dta dw  ?           ; адрес области передачи данных
        dw  ?

; Образец записи BOOT для инициализации
; первого сектора диска

boot_rec equ $

```



```

db 3 dup(0)
db 'MSDOS4.0'
dw 512
db 1
dw 1
db 2
dw 64
dw 360
db 0fch
dw 2

```

```

;=====

```

```

; Программа стратегии

```

```

dev_strategy:

```

```

    mov cs:req_seg,es
    mov cs:req_off,bx
    ret

```

```

; Здесь запоминается адрес заголовка запроса

```

```

req_seg      dw ?
req_off      dw ?

```

```

;=====

```

```

; Обработчик прерывания

```

```

dev_interrupt:

```

```

    push es      ; сохраняем регистры
    push ds
    push ax
    push bx
    push cx
    push dx
    push si
    push di
    push bp

```

```

; Устанавливаем ES:BX на заголовок запроса

```

```

    mov ax,cs:req_seg
    mov es,ax
    mov bx,cs:req_off

```

```

; Получаем код команды из заголовка запроса и умножаем
; его на два, чтобы использовать в качестве индекса
; таблицы адресов обработчиков команд

```

```

    mov al,es:[bx]+2
    shl al,1

    sub ah,ah      ; Обнуляем AH
    lea di,functions ; DI указывает на смещение
                    ; таблицы
    add di,ax      ; Добавляем смещение в таблице
    jmp word ptr [di] ; Переходим на адрес из таблицы

```


functions LABEL WORD ; Таблица функций

```
dw   initialize
dw   check_media
dw   make_bpb
dw   ioctl_in
dw   input_data
dw   nondestruct_in
dw   input_status
dw   clear_input
dw   output_data
dw   output_verify
dw   output_status
dw   clear_output
dw   ioctl_out
dw   Device_open
dw   Device_close
dw   Removable_media
```

; Выход из драйвера, если функция не поддерживается

```
ioctl_in:
nondestruct_in:
input_status:
clear_input:
output_status:
clear_output:
ioctl_out:
Removable_media:
Device_open:
Device_close:
```

```
      or    es:word ptr [bx]+3,8103h
      jmp   quit
```

; =====

; Построение блока BPB

make_bpb:

```
      push   es
      push   bx

      mov    cs:WORD PTR start_sec,0
      mov    cs:WORD PTR total,1
      call   calc_adr

      push   cs
      pop    es

      lea    di,bpb
      add    si,11
      mov    cx,13
      rep    movsb

      pop    bx
      pop    es
```



```
lea    dx,bpb
mov     es:18[bx],dx
mov     es:20[bx],cs
jmp     quit
```

check_media:

; Проверка смены носителя данных.
; Носитель не менялся.

```
mov     es:BYTE PTR 14[bx],1
jmp     quit
```

; Обработчик команды вывода данных

output_verify:

; Для вывода с проверкой устанавливаем флаг проверки

```
mov     cs:BYTE PTR verify,1
```

output_data:

```
call    in_save
mov     ax,es:WORD PTR 20[bx]
mov     cs:start_sec,ax

mov     ax,es:WORD PTR 18[bx]
mov     cs:total,ax

call    sector_write

mov     es,cs:req_seg
mov     bx,cs:req_off

cmp     cs:BYTE PTR verify,0
jz      no_verify

mov     cs:BYTE PTR verify,0
jmp     input_data
```

no_verify:

```
jmp     quit
```

;=====

; Обработчик команды ввода данных

input_data:

```
call    in_save
mov     ax,es:WORD PTR 20[bx]
mov     cs:start_sec,ax

mov     ax,es:WORD PTR 18[bx]
mov     cs:total,ax
```



```
call sector_read

mov es,cs:req_seg
mov bx,cs:req_off

jmp quit
```

```
;=====
quit:
```

```
or es:word ptr [bx]+3,100h
pop bp
pop di
pop si
pop dx
pop cx
pop bx
pop ax
pop ds
pop es
ret
```

```
;=====
; Процедура выводит на экран строку
; символов в формате ASCIIZ
```

```
dpc proc near
    push si
dpc_loop:
    cmp ds:byte ptr [si],0
    jz end_dpc
    mov al,ds:byte ptr [si]
    @@out_ch al
    inc si
    jmp dpc_loop

end_dpc:
    pop si
    ret
dpc endp
```

```
;=====
hello db 13,10,'
      db 13,10,' *RAM/DISK* (C)Frolov A., 1990
      db 13,10,'
      db 13,10,0
```

```
;=====
; Сохранение адреса буфера и значения счетчика
; из области запроса в области локальных данных
```

```
in_save proc near

    mov ax,es:WORD PTR 14[bx]
```



```
mov     cs:user_dta,ax
mov     ax,es:WORD PTR 16[bx]
mov     cs:user_dta+2,ax

mov     ax,es:WORD PTR 18[bx]
xor     ah,ah
mov     cs:total,ax

ret
```

in_save endp

; Процедура пересчитывает адрес сектора
; в адрес соответствующего этому сектору
; блока памяти. В регистре DS возвращается
; сегментный адрес этого блока,
; в CX – общее количество байт во всех секторах.
; Количество секторов задается в total,
; номер начального сектора – в start_sec

calc_adr proc near

```
mov     ax,cs:start_sec
mov     cx,20h
mul     cx

mov     dx,cs:vdisk_ptr
add     dx,ax
mov     ds,dx

xor     si,si
mov     ax,cs:total
mov     cx,512
mul     cx

or      ax,ax
jnz     move_it

mov     ax,0ffffh
```

move_it:

```
xchg    cx,ax
ret
```

calc_adr endp

; Чтение сектора из памяти виртуального диска

sector_read proc near

```
call    calc_adr
mov     es,cs:user_dta+2
mov     di,cs:user_dta

mov     ax,di
add     ax,cx
jnc     read_copy
```



```
    mov     ax,0ffffh
    sub     ax,di
    mov     cx,ax
read_copy:
    rep     movsb
    ret
```

sector_read endp

; Запись сектора в память виртуального диска

sector_write proc near

```
    call    calc_adr
    push    ds
    pop     es
    mov     di,si
    mov     ds,cs:user_dta+2
    mov     si,cs:user_dta

    mov     ax,si
    add     ax,cx
    jnc     write_copy
    mov     ax,0ffffh
    sub     ax,si
    mov     cx,ax
write_copy:
    rep     movsb
    ret
```

sector_write endp

;=====

E_O_P: ;Метка конца программы

;=====

initialize:

```
    push    cs
    pop     dx

    lea     ax,cs:vdisk ; начало памяти, в которой
                        ; расположен диск

    mov     cl,4
    ror     ax,cl
    add     dx,ax
    mov     cs:vdisk_ptr,dx

    mov     ax,2d00h     ; размер памяти, отведенной
                        ; для диска

    add     dx,ax
```

; Записываем в область запроса адрес за
; концом области памяти, отведенной диску

```
    mov     es:word ptr [bx]+14,0
    mov     es:word ptr [bx]+16,dx
```


; Количество поддерживаемых логических дисков - 1

```
mov     es:word ptr [bx]+13,1
```

; Возвращаем адрес построенного BPB

```
lea     dx,bpb_ptr
mov     es:word ptr [bx]+18,dx
mov     es:word ptr [bx]+20,cs
```

; Инициализируем BOOT-сектор

```
mov     es,cs:vdisk_ptr
xor     di,di
lea     si,boot_rec
mov     cx,24
rep     movsb
```

; Обнуляем два сектора для FAT

```
mov     cs:WORD PTR start_sec,1
mov     cs:WORD PTR total,2
call    calc_adr

push    ds
pop     es
mov     di,si
xor     al,al
rep     stosb
```

; Подготавливаем первую копию FAT

```
mov     ds:BYTE PTR [si],0feh
mov     ds:BYTE PTR 1[si],0ffh
mov     ds:BYTE PTR 2[si],0ffh
```

; Подготавливаем вторую копию FAT

```
push    ds
push    si

mov     cs:WORD PTR start_sec,3
mov     cs:WORD PTR total,2
call    calc_adr

push    ds
pop     es
mov     di,si

pop     si
pop     ds

rep     movsb
```

; Записываем нули в секторы корневого каталога

```
mov     cs:WORD PTR start_sec,5
mov     cs:WORD PTR total,4
call    calc_adr
```



```

xor    al,al
push   ds
pop     es
xor     di,di
rep     stosb

```

; Выводим сообщение

```

mov ax,cs
mov ds,ax
mov si,offset hello
call dpc

```

```

jmp quit

```

; Здесь начинается область данных, в которой
; расположен электронный диск. Эта область
; выравнена на границу параграфа.

```

ALIGN 16

```

```

vdisk equ $

```

```

ram    ENDP

```

```

END ram

```

Продолжение, начало см. стр. 35, 68, 98

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ МУЛЬТИТРАНСПЬЮТЕРНЫХ СИСТЕМ

Б.Ю. Сырков, С.В. Матвеев

Рассмотрено функционирование и использование программного обеспечения мультитранспьютерных систем. Приведены оценки производительности транспьютера; описан язык Оккам-2, распределенная ОС Гелиос; средства создания параллельных программ для мультитранспьютеров в среде Гелиос; рассмотрено программирование мультитранспьютерных систем с использованием языков Си, Паскаль, Фортран, Модула-2; изложение иллюстрировано примерами.

Объем 12 л. Ориентировочная цена 25 руб.

А. Григорьев. С для РС. ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С ДЛЯ ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРОВ

Вып. 1.	Связь С-программ с операционной системой
Вып. 2.	Время и звук
Вып. 3.	Стандартный ввод/вывод и прерывания
Вып. 4.	Графика на экране I
Вып. 5.	Графика на экране II
Вып. 6.	Текст и графика на принтере
Вып. 7.*	
Вып. 8.	Плоттер и диджитайзер
Вып. 9.	Мышь

* Внимание: вып. 7 снят с издания (нумерация последующих выпусков оставлена прежней во избежание путаницы с заказами).

Объем выпуска 3-4 л., цена - 6.50 руб.

Окончание на стр. 236

Приложение 1

ФОРМАТЫ УПРАВЛЯЮЩИХ БЛОКОВ

Заголовок драйвера

(0)	4	next	- указатель на заголовок следующего драйвера. Если смещение адреса следующего драйвера равно FFFF, это последний драйвер в цепочке
(+4)	2	attrib	- атрибуты драйвера
(+6)	2	strateg	- смещение программы стратегии драйвера
(+8)	2	interrupt	- смещение программы обработки прерывания для драйвера
(+10)	8	dev_name	- имя устройства для символьных устройств или количество обслуживаемых устройств для блочных устройств

Слово атрибутов устройства (символьные устройства)

Бит	Назначение
0	1 - драйвер обслуживает стандартное устройство ввода; 0 - этот драйвер не обслуживает стандартное устройство ввода
1	1 - драйвер обслуживает стандартное устройство вывода; 0 - драйвер не обслуживает стандартное устройство вывода
2	1 - это драйвер стандартного устройства NUL; 0 - драйвер не обслуживает устройство NUL
3	1 - драйвер обслуживает часы
4	Зарезервировано, бит должен быть равен 0
5	Зарезервировано, бит должен быть равен 0
6	1 - разрешено использование драйвером функций GENERIC IOCTL (для версий DOS, более поздних, чем 3.2); 0 - функции GENERIC IOCTL не поддерживаются
7-10	Эти биты зарезервированы и должны быть равны 0
11	1 - поддерживаются функции открытия/закрытия устройства (OPEN/CLOSE) для символьных устройств; 0 - функции OPEN/CLOSE для символьных устройств не поддерживаются
12	Зарезервировано, бит должен быть равен 0

- | | |
|----|---|
| 13 | 1 - для символьных устройств поддерживается функция вывода до получения состояния занятости устройства; |
| | 0 - функция вывода до состояния занятости не поддерживается |
| 14 | 1 - поддерживаются функции IOCTL; |
| | 0 - функции IOCTL не поддерживаются |
| 15 | 1 - символьное устройство; |
| | 0 - блочное устройство |

Слово атрибутов устройства (блочные устройства)

Бит	Назначение
0	Зарезервировано, бит должен быть равен 0
1	1 - драйвер поддерживает 32-битовую адресацию сектора (для версий DOS, начиная с 4.00 и более поздних); если установлен этот бит, поле номера сектора всех запросов является двойным словом, добавляемым в конец заголовка запроса, старое поле номера сектора должно содержать -1);
	0 - используется 16-битовая адресация сектора
2-5	Эти биты зарезервированы и должны быть равны 0
6	1 - поддерживаются логические устройства (используется блочными драйверами для управления "виртуальными" флоппи-дисками, создаваемые драйвером DRIVER.SYS в MS-DOS версии 3.2 и более поздних версиях);
	0 - логические устройства для блочных драйверов не поддерживаются
7-10	Эти биты зарезервированы и должны быть равны 0
11	1 - единица в этом бите означает, что драйвер поддерживает функцию проверки замены носителя данных в устройстве (например, замены дискеты), используется для MS-DOS версий 3.00 и более поздних;
	0 - для блочных устройств функция проверки замены носителя данных не поддерживается
12	Зарезервировано, бит должен быть равен 0
13	1 - драйвер не использует стандартное IBM-устройство и необходимо выдать запрос на построение блока параметров BIOS BPB;
	0 - используется IBM-устройство
14	1 - поддерживаются функции IOCTL;
	0 - функции IOCTL не поддерживаются
15	1 - символьное устройство;
	0 - блочное устройство

Заголовок запроса

(0)	1	size	- длина запроса в байтах (длина заголовка запроса плюс длина переменной части запроса)
(+1)	1	unit	- номер устройства (используется для блочных устройств, указывает, с каким именно устройством, обслуживаемым драйвером, будет работать операционная система)
(+2)	1	cmd	- код команды, которую требуется выполнить (может иметь значение от 0 до 18h)
(+3)	2	status	- слово состояния устройства, заполняется драйвером перед возвратом управления операционной системе
(+5)	8	reserved	- зарезервировано

Слово состояния устройства

Бит	Назначение
0-7	Код ошибки устройства (если команда выполнялась с ошибкой и драйвер установил признак ошибки (бит 15) в единицу, то в это поле он должен записать код ошибки)
8	Команда выполнена. Этот бит всегда устанавливается драйвером перед тем, как он возвращает управление операционной системе
9	Занято. Этот бит устанавливается обработчиком команды, когда физическое устройство занято выполнением предыдущей операции и поэтому не может выполнить требуемую команду. Этот бит используется также для передачи такой информации, как "буфер клавиатуры не пуст", "среда носителя данных заменяемая" (в команде проверки возможности замены среды носителя данных)
10-14	Зарезервировано
15	Признак ошибки. Устанавливается драйвером, когда он не может обработать запрос или произошла физическая либо логическая ошибка при обработке правильного запроса. Биты 0 - 7 при этом должны содержать код ошибки

Коды ошибок

<i>Код</i>	<i>Описание</i>
0	Нарушение защиты от записи. Была предпринята попытка записи информации на защищенное от записи устройство
1	Неизвестное устройство
2	Устройство не готово
3	Неизвестная команда. Затребованная команда не поддерживается драйвером
4	Ошибка CRC. При выполнении команды обнаружена ошибка циклического кода проверки
5	Неправильная длина запроса. Поле длины в заголовке запроса содержит неверное значение
6	Ошибка при поиске дорожки (дорожка не найдена)
7	Неизвестный носитель данных
8	Сектор не найден
9	Нет бумаги в принтере
0Ah	Ошибка записи
0Bh	Ошибка чтения
0Ch	Общая ошибка
0Dh	Зарезервировано
0Eh	Зарезервировано
0Fh	Неразрешенная замена диска (только для MS-DOS версии 3.0 и более поздних версий)

Блок BPB

(0)	2	sect_siz	- количество байтов в одном секторе диска
(+2)	1	clustsiz	- количество секторов в одном кластере
(+3)	2	res_sect	- количество зарезервированных секторов
(+5)	1	fat_cnt	- количество таблиц FAT
(+6)	2	root_siz	- максимальное количество дескрипторов файлов, содержащихся в корневом каталоге диска
(+8)	2	tot_sect	- общее количество секторов на носителе данных (в разделе DOS)
(+10)	1	media	- байт-описатель среды носителя данных
(+11)	2	fat_size	- количество секторов, занимаемых копией FAT

Форматы запросов для различных команд

0 - Инициализация

(0)	13	header	- заголовок запроса
(+13)	1	n_units	- количество устройств, обслуживаемых драйвером. Это поле заполняется только блочным драйвером
(+14)	4	end_addr	- конечный FAR-адрес резидентной части кода драйвера. В это поле драйвер записывает адрес байта памяти, следующего за той частью кода драйвера, которая должна стать резидентной
(+18)	4	parm	- FAR-адрес строки параметров инициализации драйвера из файла CONFIG.SYS. Эта строка содержит все, что находится в строке файла после команды 'DEVICE=', и заканчивается символами перевода строки и возврата каретки 0Ah, 0Dh. При возврате драйвер блочного устройства должен записать в это поле адрес массива указателей на блоки параметров BIOS (BPB), по одному указателю на каждое обслуживаемое драйвером устройство
(+22)	1	drive	- номер устройства. Для MS-DOS версии 3.0 и более поздних в это поле при загрузке драйвера операционная система заносит номер, назначенный устройству, обслуживаемому драйвером. Например, для устройства A это 0, для B - 1 и т. д.

1 - Проверка замены носителя

(0)	13	header	- заголовок запроса
(+13)	1	media	- в этом поле драйверу передается байт-описатель среды носителя данных, с которым DOS работала раньше
(+14)	1	reply	- в это поле драйвер должен поместить ответ о факте замены среды: 1 - диск не заменялся; 0 - неизвестно; -1 - диск был заменен
(+15)	4	vol_id	- указатель на предыдущую метку тома (если установлен бит 11 слова атрибута устройства и диск был заменен)

2 - Построить блок BPB

(0)	13	header	- заголовок запроса
(+13)	1	media	- в этом поле драйверу передается байт-описатель среды носителя данных, с которым DOS работала раньше
(+14)	4	buf_adr	- адрес буфера обмена. Содержимое этого буфера при вызове драйвера зависит от установки бита 13 слова атрибутов устройства. (IBM-формат). Если этот бит равен 0 (устройство формата IBM), буфер содержит первый сектор первой копии FAT. В противном случае указатель установлен на буфер свободного сектора
(+18)	4	bpb_adr	- указатель на новый BPB, записывается в это поле драйвером

3, 4, 8, 9, 0Ch - Чтение/запись

(0)	13	header	- заголовок запроса
(+13)	1	media	- в этом поле драйверу передается байт-описатель среды носителя данных
(+14)	4	buf_adr	- адрес буфера для передачи данных
(+18)	2	count	- количество передаваемых байтов для символьных устройств или секторов для блочных устройств
(+20)	2	sector	- номер начального сектора, если драйвер использует 16-битовую адресацию секторов или -1 для 32-битовой адресации. Это поле не используется символьными драйверами
(+22)	4	vol_id	- указатель на метку тома в формате ASCIIZ. Возвращается блочным драйвером, если он дает ошибку 15 (неправильная смена диска). Это поле должно содержать ссылку на метку требуемого диска
(+26)	4	sect32	- номер начального сектора, если содержимое поля sector равно -1. Первым идет старшее слово номера сектора. Если обнаружена ошибка с номером 15, в это поле записывается указатель на метку тома

5 - Неразрушающий ввод

(0)	13	header	- заголовок запроса
(+13)	1	byte	- в это поле драйвер записывает извлеченный из буфера байт, который будет считан по следующей команде ввода

6, 0Ah - Проверит состояние ввода/вывода

Для команд проверки состояния запрос состоит только из заголовка, область переменного формата отсутствует.

7 - Сброс буфера устройства ввода

11 - Сброс буфера устройства вывода

Запрос состоит только из заголовка.

0Dh, 0Eh - Открыть/закрыть устройство

Запрос для этих команд состоит только из заголовка.

15 - Проверка сменяемости диска

Запрос состоит только из заголовка.

19 - Общее управление вводом/выводом (GENERIC IOCTL)

(0)	13	header	- заголовок запроса
(+13)	1	funct	- это поле содержит код функции команды общего IOCTL
(+14)	1	subfunc	- код подфункции для функции funct
(+15)	2	si_reg	- значение регистра SI при вызове функции 44h прерывания 21h. Эта функция DOS предназначена для управления вводом/выводом
(+17)	2	di_reg	- значение, передаваемое при вызове функции 44h прерывания 21h через регистр DI
(+19)	4	buf	- указатель на буфер данных, содержащий управляющую информацию для устройства или предназначенный для приема информации от устройства

23 - Получить активное логическое устройство

24 - Установить активное логическое устройство

(0)	13	header	- заголовок запроса
(+13)	1	unit	- код логического устройства, которое должно стать активным при использовании команды 24, или код активного устройства, помещаемый драйвером по команде 23
(+14)	1	cmd	- код команды, как используется - неизвестно
(+15)	4	status	- слово состояния, формат его неизвестен
(+19)	4	reserved	- зарезервировано

Приложение 2

КОМАНДЫ ДРАЙВЕРА

- 0 - Инициализация
- 1 - Проверка смены носителя данных
- 2 - Построить блок BPB
- 3 - Чтение IOCTL
- 4 - Чтение
- 5 - Неразрушающее чтение
- 6 - Проверка состояния ввода
- 7 - Сброс буферов ввода
- 8 - Запись
- 9 - Запись с проверкой
- 10 - Проверка состояния вывода
- 11 - Сброс буферов вывода
- 12 - Запись IOCTL
- 13 - Открыть устройство
- 14 - Закрыть устройство
- 15 - Проверка возможности смены носителя данных
- 16 - Зарезервировано
- 17 - Зарезервировано
- 18 - Зарезервировано
- 19 - Общее управление вводом/выводом (GENERIC IOCTL)
- 20 - Получить текущее логическое устройство
- 21 - Установить логическое устройство

СЕГМЕНТНАЯ АДРЕСАЦИЯ ПАМЯТИ

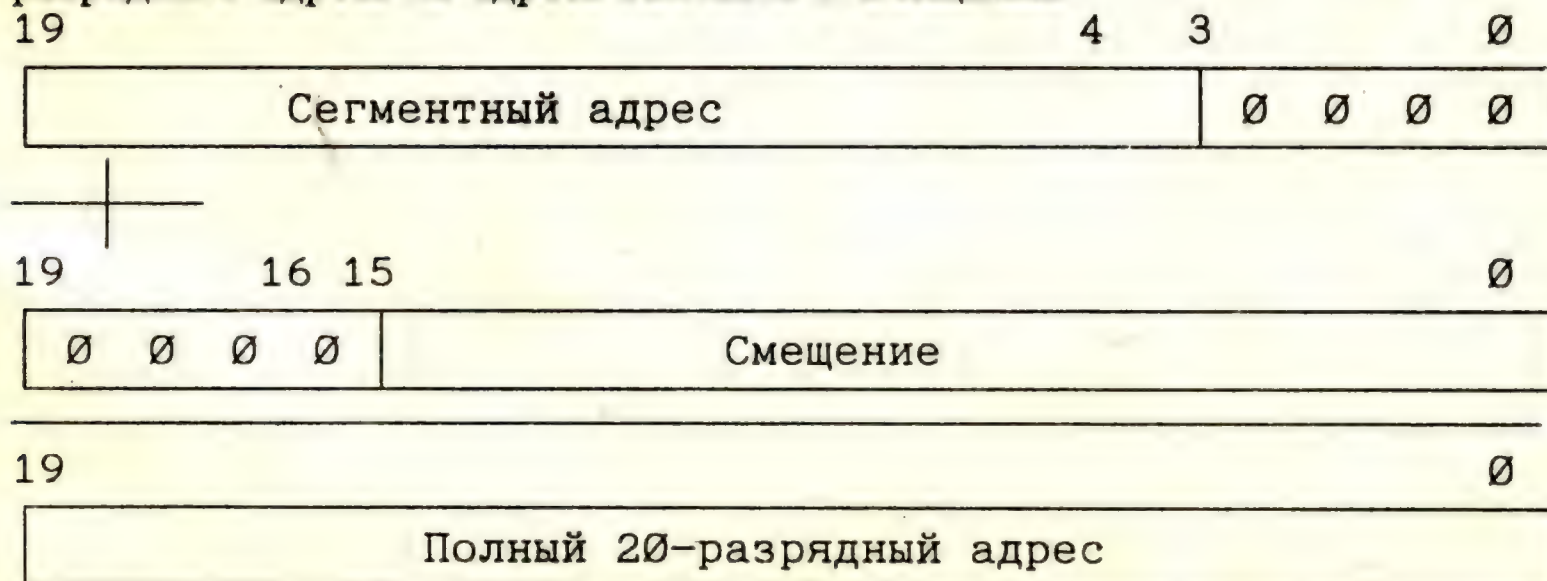
Особенностью архитектуры процессоров INTEL 8086, 80286, 80386, 80486 является использование механизма сегментации адресного пространства. Сегментация вызывает трудности у тех программистов, которые раньше работали на ЭВМ типов PDP, CM ЭВМ, ЕС ЭВМ. В этих машинах программа имеет дело с логическими адресами, которые тем или иным способом отображаются на физические адреса. Программа может не знать подробностей отображения логических адресов на физические, она работает только с логическим адресом.

Прообраз процессора 8086 - оригинальный микропроцессор INTEL 8080 - имел линейное адресное пространство размером 64 Кбайта. В этом микропроцессоре логический и физический адреса совпадали - все 16 адресных линий (адресных шин) использовались непосредственно для адресации памяти, а программы оперировали абсолютными шестнадцатиразрядными адресами.

Однако быстро растущие потребности программ в оперативной памяти привели к необходимости расширения адресного пространства. Следующий микропроцессор 8086 имел уже 20 адресных линий, что позволило непосредственно адресовать до 1 Мбайта оперативной памяти. Архитектурное решение этого микропроцессора позволило легко адаптировать накопленное в большом количестве программное обеспечение для микропроцессора 8080. Микропроцессор 8086 является 16разрядным, поэтому использование 20разрядного адреса в 16-разрядных командах неэффективно. Вместо указания в командах полного 20-разрядного адреса используется двухкомпонентная адресация, причем каждая компонента использует только 16 разрядов.

Эти компоненты называются сегментной компонентой адреса и компонентой смещения. Логический 20-разрядный адрес получается сложением двух компонент, причем сегментный адрес перед сложением умножается на 16 (сдвигается влево на 4 разряда). Сложение и сдвиг выполняется аппаратно, поэтому на формирование 20-разрядного адреса дополнительно время не затрачивается.

На рисунке показано, как в процессоре 8086 происходит формирование 20-разрядного адреса из адреса сегмента и смещения:



Адрес сегмента сдвигается влево на 4 бита с заполнением младших битов нулями, смещение расширяется до 20 битов и складывается со сдвинутым адресом сегмента. Например, если адрес сегмента равен 1234h, смещение равно 1116h, то полный 20-разрядный адрес будет $12340h + 01116h = 13456h$.

Таким образом, оперируя 16-разрядными адресами сегмента и смещением, процессор может адресовать 1 Мбайт памяти. Для хранения сегментных адресов и смещений процессор имеет специальные регистры.

Каждая выполняющаяся программа в любой момент времени может адресоваться сразу к четырем сегментам памяти. Это сегмент кода, сегмент данных, дополнительный сегмент данных, сегмент стека. Сегмент кода содержит выполняющиеся машинные команды, сегменты данных и дополнительных данных используются для размещения используемых программой переменных, массивов и других структур данных, сегмент стека используется при вызове подпрограмм. На рисунке показан вариант размещения сегментов в оперативной памяти:



Сегменты могут перекрываться или не перекрываться. На рисунке показаны неперекрывающиеся сегменты.

Для хранения сегментных адресов процессор имеет 4 сегментных регистра: CS, DS, ES, SS. Эти регистры содержат соответственно адреса сегментов кода, данных, дополнительных данных и стека.

При адресации выполняющегося кода вместе с регистром CS используется регистр смещения IP. Пара регистров CS:IP всегда указывает на текущую выполняющуюся команду.

Адресация данных возможна относительно любого сегментного регистра. При этом смещение может указываться как непосредственно в команде, так и с помо-

щью регистров. Программа должна сама следить за правильной загрузкой и использованием сегментных регистров.

Мы приведем несколько примеров программ, составленных на языке ассемблера. Эти программы используют различное количество сегментов и могут служить шаблоном для составления ваших собственных программ.

Первая программа использует всего один сегмент. В этом сегменте расположены выполняющиеся машинные команды и данные, используемые программой. Заметьте, что размер программы, состоящей из одного сегмента, не может превышать 64 Кбайта.

Текст программы:

```
; В этом месте расположен сегмент кода. Он содержит
; выполняющуюся программу.
code    segment

; Директива assume сообщает ассемблеру, как будут
; использоваться сегментные регистры. Эта директива
; не выполняет загрузку сегментных регистров, она нужна
; ассемблеру только для правильного вычисления смещений.
        assume cs:code, ds:code

; Эта строка нужна для создания com-программы.
        org 100h

; При запуске программы управление будет передано на оператор
; с меткой start. Первое, что должна сделать программа -
; правильно загрузить сегментные регистры. Регистр CS
; загружается операционной системой при запуске программы,
; поэтому его загружать не надо. Регистры DS и ES должны
; указывать на начало сегмента кода, т. к. программа
; состоит из одного сегмента.
start:
        mov     ax, cs
        mov     ds, ax

; Выводим сообщение msg из сегмента данных
        mov     ah, 9h
        mov     dx, OFFSET msg
        int     21h

; Завершаем работу программы
        mov     ax, 4C00h
        int     21h

; Строка, которую программа выведет на экран.
msg      db      "Hello, world.", 13, 10, "$"
code     ends

        end     start
```


Если для размещения данных и буферов недостаточно одного сегмента, необходимо организовать отдельные сегменты для кода и данных, как это сделано в следующем примере:

```
; Создаем сегмент стека. Размер стека – 256 байт,  
; стек выравнен на границу параграфа (para).  
stack    segment para stack  
  
; Резервируем 256 байт для стека.  
        db      100h dup (?)  
  
stack    ends  
  
; Создаем сегмент данных. Этот сегмент выравнен на  
; границу двухбайтового слова (word).  
data     segment word  
  
; Строка, которую программа выведет на экран.  
msg      db      "Hello, world.", 13, 10, "$"  
  
data     ends  
  
; В этом месте расположен сегмент кода. Он содержит  
; выполняющуюся программу.  
code     segment  
  
; Директива assume сообщает ассемблеру, как будут  
; использоваться сегментные регистры. Эта директива  
; не выполняет загрузку сегментных регистров, она  
; нужна ассемблеру только для правильного вычисления  
; смещений.  
        assume   cs:code, ds:data, ss:stack  
  
; При запуске программы управление будет передано  
; на оператор с меткой start.  
; Первое, что должна сделать программа – правильно  
; загрузить сегментные регистры.  
;  
; Следующие два оператора инициализируют сегментный  
; регистр данных DS.  
start:  
        mov      ax, data  
        mov      ds, ax  
  
; Инициализируем сегментный регистр стека и  
; указатель стека (регистры SS и SP).  
; Эта операция должна выполняться в состоянии  
; процессора с запрещенными прерываниями, иначе,  
; если регистр SS будет содержать уже новое значение,  
; а SP – старое и если в этот момент произойдет
```


; прерывание, адрес возврата и значение регистра флагов
; будут записаны в не предназначенную для этого область.

```
cli
mov     ss, ax
mov     sp, OFFSET stack
sti
```

; Выводим сообщение msg из сегмента данных

```
mov     ah, 9h
mov     dx, OFFSET msg
int     21h
```

; Завершаем работу программы

```
mov     ax, 4C00h
int     21h
```

code ends

end start

Макроассемблер MASM версии 5.0 и более поздних версий, а также QuickAssembler содержит директивы, упрощающие описание сегментов. Это такие директивы, как .CODE, .DATA, .MODEL и др. Вы найдете подробное описание этих директив в соответствующей документации по ассемблеру. Остановимся подробнее на директиве .MODEL.

Эта директива задает так называемую модель памяти, используемую программой. Что это такое?

Мы уже говорили о том, что программа должна состоять из одного или нескольких сегментов, в зависимости от размера кода и данных, которыми она оперирует. Существует несколько стандартных вариантов использования сегментов, которые называются моделями памяти. Используется шесть таких моделей:

- Tiny
- Small
- Medium
- Compact
- Large
- Huge

Модель памяти Tiny используется небольшими программами, состоящими из одного сегмента и имеющими формат COM. Использование этой модели памяти - единственный способ получения загрузочного модуля в формате COM.

В модели Small один сегмент используется для кода, один для хранения данных и размещения стека программы. Общий размер программы в этом случае ограничен величиной 128 Кбайт. Большинство небольших программ используют именно эту модель памяти.

Если ваша программа оперирует небольшим объемом данных, но размер кода превышает 64 Кбайта, вам подходит модель Medium. В этой модели используется несколько сегментов для хранения кода и только один - для данных.

Модель Compact, в отличие от Medium, использует один сегмент для кода и несколько - для данных. Эта модель больше всего подходит для небольших программ, обрабатывающих большие массивы данных.

Модель памяти Large предоставляет возможность использовать несколько сегментов для кода и несколько сегментов для данных. Эта модель обычно используется большими программами, которые обрабатывают большие объемы данных.

И наконец, модель памяти Huge. Эта модель аналогична Large, но для программ, составленных на языке Си, она позволяет использовать массивы данных, имеющие размер более одного сегмента.

Приведем два примера использования моделей памяти в программах, составленных на языке ассемблера. Эти примеры аналогичны тем, которые мы только что рассмотрели. Первая программа использует модель памяти Tiny:

```
; Определяем используемую модель памяти
.model tiny

; Определяем сегмент данных.
.data
msg db "Hello, world.", 13, 10, "$"

; Определяем сегмент кода.
.code

; Макрокоманда startup выполняет все необходимые
; инициализирующие действия, которые зависят от
; модели памяти.
.startup

; Выводим сообщение на экран
mov ah, 9h
mov dx, OFFSET msg
int 21h

; Завершаем выполнение программы
.exit
END
```

Вторая программа использует модель памяти Small, в ней мы дополнительно определили свой стек:

```
; Определяем используемую модель памяти
.model small

; Определяем сегмент данных.
.data
```



```
msg      db      "Hello, world.", 13, 10, "$"
; Определяем свой стек, его размер – 256 байтов
      .stack 100h
; Определяем сегмент кода.
      .code
      .startup
; Выводим сообщение на экран
      mov     ah, 9h
      mov     dx, OFFSET msg
      int     21h
; Завершаем выполнение программы
      .exit
      END
```

Для программ, составленных на языке Си, модель памяти указывается при трансляции. Если используется пакетный транслятор, модель указывается при помощи опций в командной строке. Если вы работаете в интегрированной среде, такой, как QuickC, модель задается при помощи соответствующего меню, а сама программа не содержит каких-либо директив, определяющих используемую модель памяти.

Как правильно выбрать модель памяти?

Если ваша программа небольшая по размеру, то вам подойдут модели Tiny или Small. При использовании остальных моделей памяти возможно увеличение размера загрузочного модуля и времени выполнения программы из-за того, что в операциях с данными и при вызове подпрограмм используются полные адреса, состоящие из сегмента и смещения. Это означает, в частности, что если при трансляции программы была использована модель Large, то при обращении к каждой переменной и при вызове каждой подпрограммы (функции) будет использоваться полный адрес.

Для сокращения накладных расходов отдельные переменные и функции можно разместить в отдельном сегменте. При этом их надо описать специальным образом, используя ключевое слово `near` (для Microsoft C 6.0 и QuickC 2.5 можно использовать `_near`). Ключевое слово `near` (`_near`) сообщает транслятору, что данные должны быть размещены в некотором общем сегменте данных и доступ к ним должен осуществляться с использованием 16-битового адреса (только компонента смещения). Если с этим ключевым словом описана функция, то транслятор поместит ее в текущий сегмент кода, для вызова функции будет также использован 16-битовый адрес.

В противоположность только что описанному ключевое слово `far` (`_far` для Microsoft C 6.0 и QuickC 2.5) говорит о том, что данные или функция могут располагаться в любом месте памяти, не обязательно в текущем сегменте, и для адресации необходимо использовать полный 32-битовый адрес.

Ключевое слово `huge` (`_huge`) необходимо использовать при описании массивов, которые по своим размерам могут превышать 64 Кбайта. Для адресации при этом будет использоваться полный 32-битовый адрес. Для функций это ключевое слово не применяется.

Приведем несколько примеров описания данных и функций с использованием ключевых слов `near`, `far`, `huge`.

// Используемая модель памяти - Small

```
char dim1[250];
char _far dim2[45000];
char _huge dim3[80000];
char _far *far_ptr;
char _far * _far * far_ptr1;
```

```
int _far function1(void);
```

// Используемая модель памяти - Large

```
char _near dim4[2000];
```

```
char _far * _near function2(void);
```

Исчерпывающие сведения об использовании моделей памяти содержатся в документации на используемый транслятор.

Окончание, начало см. стр. 35, 68, 98, 220

ПРОГРАММИРОВАНИЕ В MICROSOFT WINDOWS, в 2-х частях

С.А. Гладков, Г.В. Фролов

Учебно-справочное пособие по созданию программ ("приложений"), работающих под управлением операционной оболочки MS-Windows 2.x, 3.x. Рассмотрено положение MS-Windows среди системного программного обеспечения, ее внутренняя структура. Обсуждаются вопросы установки среды программирования, взаимодействие приложения с операционной оболочкой, алгоритм создания программы. Раскрываются аспекты программирования в Windows: структура программы, работа с экраном, клавиатурой, мышью, графика, шрифты и пр. Предметный указатель позволяет использовать книгу как справочник функций MS-Windows.

Объем каждой части 15 л. Ориентировочная цена каждой части 40 руб.

Все цены приведены по состоянию на 12.02.92

Открытку с предварительным заказом

Вы можете отправить по адресу:

115409 Москва, ул. Москворечье, 31, корп. 2.

Заказы выполняются в порядке поступления заявок по мере выхода изданий из печати. Книги высылаются наложенным платежом (стоимость пересылки одной бандероли - 25 руб.).

Жителям Москвы, а также всем желающим выкупать заказанную литературу непосредственно в нашем киоске необходимо оформлять заказ на каждую книгу на отдельной открытке со своим заполненным адресом.

Телефоны для справок о наличии литературы в продаже и выполнении заказов: 320-43-77, 320-31-88

ОГЛАВЛЕНИЕ

От издательства.....	3
Предисловие авторов	4

Книга 1

Глава 1. ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ MS-DOS	7
1.1. Основные компоненты операционной системы	7
1.2. Процесс загрузки операционной системы	9
1.3. Общая схема работы DOS	10
Файловая система	11
Система управления памятью	12
Система управления программами	12
Система связи с драйверами устройств	12
Система обработки ошибок	13
Служба времени	13
Ввод/вывод на консоль оператора	14
Другие подсистемы DOS	14
1.4. Обзор прерываний BIOS	15
Получение информации об оборудовании	16
INT 11h - Получить список оборудования	16
INT 12h - Получить размер основной памяти	17
Работа с дисплейным адаптером..... <i>INT 10h</i>	17
00h - Задание видеорежима	18
01h - Установить характеристики курсора	18
02h - Установить положение курсора	18
03h - Получить положение курсора	18
04h - Получить положение светового пера	18
05h - Выбрать активную страницу дисплейной памяти	18
06h, 07h - Прокрутка (скроллинг) окна вверх/вниз	19
08h, 09h - Прочитать/записать символ и атрибут	19
0Ah - Записать символ	19
0Bh - Выбрать цветовую палитру	19
0Ch, 0Dh - Записать/прочитать графическую точку	19
0Eh - Запись символа на экран в стиле TTY	19
0Fh - Прочитать текущий видеорежим	19
10h, 11h, 12h - Обслуживание адаптера EGA	20
13h - Запись строки	20
Обслуживание клавиатуры..... <i>INT 16h</i>	20
Обслуживание дисковой подсистемы..... <i>INT 13h</i>	20
00h - Сброс дисковой системы	21
01h - Получить состояние дисковой системы	21
02h/03h - Чтение/запись секторов	21
04h - Проверка секторов	21
Другие функции прерывания INT 13h	21
Вывод на принтер (параллельный порт)	21
Обслуживание последовательного порта связи..... <i>INT 14h</i>	21
Работа с системными часами	22
Перезагрузка операционной системы	22
Расширенный сервис для машин класса AT	22
1.5. Обзор прерываний DOS	22
Получение системной информации	23
Символьный ввод/вывод	24

Работа с файловой системой.....	25
Управление программами	25
Управление памятью	25
Связь с драйверами устройств.....	26
Другие функции	26
1.6. Обработка ошибок.....	26
Глава 2. ВЕКТОРНАЯ ТАБЛИЦА СВЯЗИ DOS	36
2.1. Таблица связи управляющих блоков MS-DOS.....	36
2.2. Блоки управления памятью в MS-DOS.....	41
2.3. Список управляющих блоков устройств	49
2.4. Таблица файлов MS-DOS	56
2.5. Список загружаемых драйверов устройств	62
2.6. Другие поля векторной таблицы связи	66
Глава 3. УПРАВЛЕНИЕ ПРОГРАММАМИ	69
3.1. Форматы программных файлов	69
3.2. Процесс загрузки программ в память	74
3.3. Префикс программного сегмента.....	75
3.4. Запуск программ из программ.....	79
3.5. Завершение работы программы.....	86
Глава 4. ПРЕРЫВАНИЯ	87
4.1. Механизм прерываний	87
4.2. Таблица векторов прерываний	88
4.3. Маскирование прерываний	91
4.4. Изменение таблицы векторов прерываний	91
4.5. Особенности обработки аппаратных прерываний.....	95
Глава 5. РЕЗИДЕНТНЫЕ ПРОГРАММЫ	99
5.1. Что такое резидентная программа?	99
5.2. Как программе стать резидентной.....	100
5.3. Вызов резидентной программы.....	100
5.4. Особенности резидентных программ.....	101
5.5. Примеры резидентных программ.....	103
Приложение 1. УПРАВЛЯЮЩИЕ БЛОКИ	116
Векторная таблица связи MS-DOS:.....	116
Блок управления памятью MCB:	116
Префикс программного сегмента PSP:.....	117
Формат блока DDCB для MS-DOS версий 2.x и 3.x:	118
Формат блока DDCB для MS-DOS версии 4.x :.....	119
Формат таблицы файлов для DOS 3.x:	120
Формат таблицы файлов для DOS 4.x:	121
Заголовок драйвера:.....	122
Формат элемента массива дисковой информации:.....	122
Заголовок EXE-файла:.....	122
Приложение 2. СОДЕРЖИМОЕ ФАЙЛА sysp.h	124
Приложение 3. КОНТРОЛЛЕР ПРЕРЫВАНИЙ 8259	128
 Книга 2	
Глава 1. ИНТЕРФЕЙС МЕЖДУ ПРОГРАММНЫМ ОБЕСПЕЧЕНИЕМ И АППАРАТУРОЙ	132
Глава 2. СТРУКТУРА ЗАГРУЖАЕМОГО ДРАЙВЕРА.....	134

Глава 3.	ПРОЦЕСС ЗАГРУЗКИ ДРАЙВЕРОВ.....	139
Глава 4.	СВЯЗЬ ДРАЙВЕРА С ОПЕРАЦИОННОЙ СИСТЕМОЙ	141
Глава 5.	ФУНКЦИИ ЗАГРУЖАЕМОГО ДРАЙВЕРА	146
	0 - Инициализация драйвера.....	146
	1 - Проверка замены носителя данных	153
	2 - Построить блок ВРВ.....	155
	3 - IOCTL чтение, 4 - Чтение, 8 - Запись	
	9 - Запись с проверкой, 12 - IOCTL запись	
	16 - Вывод, пока не занято	155
	5 - Неразрушающее чтение без ожидания	162
	6 - Проверить состояние устройства ввода	
	10 - Проверить состояние устройства вывода.....	163
	7 - Сброс буфера устройства ввода	
	11 - Сброс буфера устройства вывода.....	163
	13 - Открыть устройство, 14 - Закрыть устройство.....	163
	15 - Проверка сменяемости диска	164
	19 - Функции общего управления вводом/выводом (IOCTL)	164
	23 - Получить активное логическое устройство	
	24 - Установить активное логическое устройство.....	165
Глава 6.	ФУНКЦИИ УПРАВЛЕНИЯ УСТРОЙСТВАМИ ВВОДА/ВЫВОДА IOCTL.....	166
Глава 7.	ДРАЙВЕР СИСТЕМНЫХ ЧАСОВ CLOCK\$	182
Глава 8.	ОСОБЕННОСТИ ОТЛАДКИ ДРАЙВЕРОВ	185
Глава 9.	ПРИМЕР ДРАЙВЕРА СИМВОЛЬНОГО УСТРОЙСТВА.....	190
Глава 10.	ПРИМЕР ДРАЙВЕРА БЛОЧНОГО УСТРОЙСТВА.....	211
Приложение 1.	ФОРМАТЫ УПРАВЛЯЮЩИХ БЛОКОВ	221
	Заголовок драйвера.....	221
	Слово атрибутов устройства (символьные устройства)	221
	Слово атрибутов устройства (блочные устройства)	222
	Заголовок запроса.....	223
	Слово состояния устройства.....	223
	Коды ошибок.....	224
	Блок ВРВ.....	224
	Форматы запросов для различных команд.....	225
	0 - Инициализация.....	225
	1 - Проверка замены носителя.....	225
	2 - Построить блок ВРВ.....	226
	3, 4, 8, 9, 0Ch - Чтение/запись.....	226
	5 - Неразрушающий ввод.....	226
	6, 0Ah - Проверит состояние ввода/вывода.....	227
	7 - Сброс буфера устройства ввода	
	11 - Сброс буфера устройства вывода.....	227
	0Dh, 0Eh - Открыть/закрыть устройство	227
	15 - Проверка сменяемости диска	227
	19 - Общее управление вводом/выводом (GENERIC IOCTL)	227
	23 - Получить активное логическое устройство	
	24 - Установить активное логическое устройство.....	227
Приложение 2.	КОМАНДЫ ДРАЙВЕРА	228
Приложение 3.	СЕГМЕНТНАЯ АДРЕСАЦИЯ ПАМЯТИ	229

"ДИАЛОГ-МИФИ" предлагает

КУРСЫ УСКОРЕННОЙ ПОДГОТОВКИ К ПРАКТИЧЕСКОЙ РАБОТЕ НА ПЕРСОНАЛЬНЫХ КОМПЬЮТЕРАХ IBM PC

УСЛОВИЯ ПРИЕМА И ОБУЧЕНИЯ

- На курсы ускоренной подготовки к практической работе на персональных компьютерах семейства IBM PC в авторизованном Учебном Центре фирм BORLAND, SYMANTEC, AUTODESK акционерного общества "ДИАЛОГ-МИФИ" принимаются как граждане СССР, так и иностранные граждане без каких-либо ограничений.
- Для обучения на курсах предварительной подготовки в области вычислительной техники, как правило, не требуется.
- Каждый день занятий - это трехчасовая лекция и пять часов индивидуальной практической работы на компьютерах IBM PC.
- Продолжительность занятий: две недели с отрывом от производства (занятия - 5 раз в неделю). Занятия проводятся под руководством высококвалифицированных преподавателей Московского инженерно-физического института и Центра (1 преподаватель - 4 слушателя).
- Каждому слушателю выдаются оригинальное пособие по изучаемому курсу, подготовленное Учебным Центром, и дискета для работы (стоимости пособия и дискеты входят в стоимость обучения).
- Иногородним предоставляется жилье.
- Стоимость обучения (на 21.01.92):
для двухнедельных курсов
- 3500 р. + 28% (кроме "Автокад 10" и "MS-DOS для программистов") без оплаты проживания и 3900 р. + 28% с оплатой;
- стоимость обучения по курсам "Автокад 10" и "MS-DOS для программистов" 4500 р. + 28% и 4900 р. + 28% соответственно;
для однонедельных курсов стоимость обучения 1750 р. + 28% без оплаты проживания и 1950 р. + 28% с оплатой.
- Оплата производится по наличному и безналичному расчетам.

- Зачисление на курсы слушателей, направляемых организацией, проводится по гарантийным письмам.
- Окончившим курсы выдается удостоверение-сертификат.

ОБУЧЕНИЕ ПРОВОДИТСЯ ПО СЛЕДУЮЩИМ КУРСАМ:

ДВУХНЕДЕЛЬНЫЕ КУРСЫ

- Практическая работа на компьютерах семейства IBM PC в операционной среде MS-DOS.
- MS-DOS для программистов (использование функций DOS и BIOS при программировании на Ассемблере).
- Программирование на Turbo Pascal.
- Программирование на Turbo C.
- Программирование на Turbo C++.
- Программирование на Microsoft Pascal.
- Программирование на Microsoft QuickPascal.
- Программирование на Microsoft QuickC.
- Программирование на Microsoft QuickBASIC.
- Практическая работа с БД Paradox.
- Практическая работа с электронной таблицей Quattro Pro.
- Практическая работа с интегрированным пакетом Reflex 2.0.
- Использование САПР Автокад 10.
- Использование электронных таблиц с деловой графикой и БД в системе Microsoft Excel (с элементами Windows).
- Локальные и распределенные вычислительные сети.
- Использование информационных технологий в медицине.

ОДНОНЕДЕЛЬНЫЕ КУРСЫ

- Практическая работа с интегрированным пакетом Microsoft Works.
- Практическая работа с текстовым процессором Microsoft Word.
- Работа в среде Microsoft Windows.

ДИАЛОГ-МИФИ
АКЦИОНЕРНОЕ ОБЩЕСТВО

СССР, 115409 Москва, ул. Москворечье, 31-2
Телефон: (095) 324-83-56, 320-30-88, 324-71-66
Телефакс: (095) 3243055

1688

ДИАЛОГ МИСД

- ОБУЧЕНИЕ** в учебном Центре, имеющем сертификаты фирм Borland, Symantec, Autodesk:
- работе на персональных компьютерах IBM PC
 - работе с новейшим прикладным программным обеспечением
 - языкам программирования
- СОЗДАНИЕ**
- программного обеспечения различного назначения
 - прикладных пакетов программ по заказам организаций
- ПРОДАЖА** через собственную широкую дилерскую сеть:
- программных продуктов ведущих зарубежных фирм: Symantec, Borland, Microsoft, Autodesk и др.
 - компьютерной техники фирм Summit Systems, Hewlett Packard и др.
 - оргтехники
- ИЗДАНИЕ**
- оригинальных книг и учебных пособий в области новых информационных технологий
- ВЫПУСК**
- товаров народного потребления
- РЕКЛАМА**
- организация выставок, презентаций
 - размещение рекламы в собственных изданиях
- СЕРВИС**
- гарантийное и послегарантийное обслуживание вычислительной техники
 - консультации в области программирования
 - сопровождение и поддержка программного обеспечения, включая hot line
 - предоставление информации о новых программных продуктах, версиях и помощь в их внедрении

Наш адрес: 115409 Москва, ул. Москворечье, 31, корп. 2

Телефоны учебного центра: 324-83-56, 320-30-88

Телефоны коммерческого отдела: 320-32-11, 320-43-44

Телефоны издательского отдела: 320-43-77

Телефакс: (095) 3243055